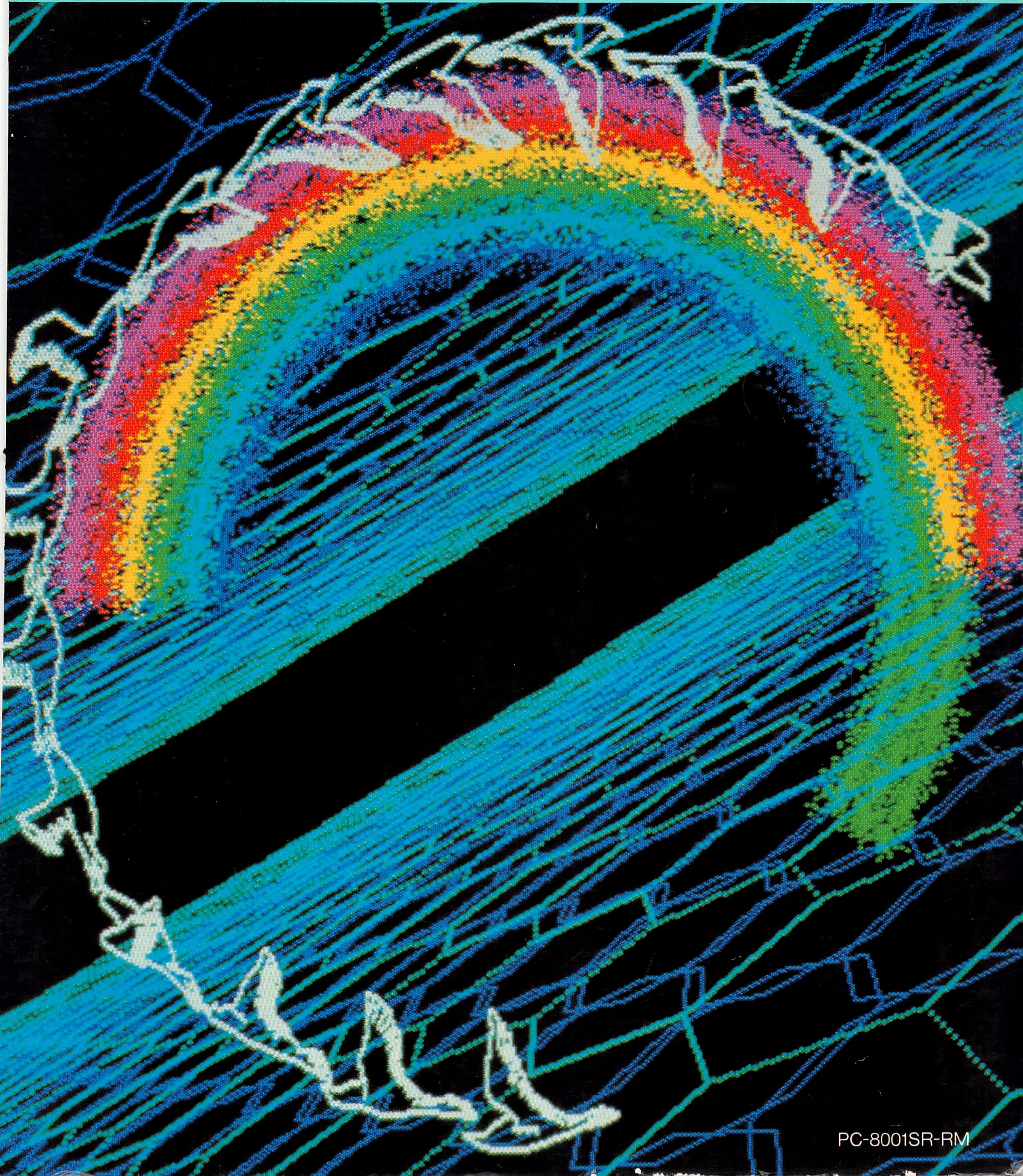
NEC Personal Computer

PC-8001 mkII SR

N80SR-BASIC

REFERENCE MANUAL



PC-8001SR-RM

PC-8001mkIISR

N80SR-BASIC

REFERENCE MANUAL

まえがき

N_{80SR}-BASICは，パーソナルコンピュータPC-8001MKⅡSRの主プログラミング言語です．

本書は，このN_{80SR}-BASICの命令の個々について，詳細に解説したリファレンスマニュアルです．

さらに，PC-8001MKⅡSRは，従来機種との互換性を保つために，N_{80}-BASIC, N-BASICを備えています．この2つのBASICについても，資料で解説されています．

このマニュアルを読む前に，PC-8001MKⅡSRユーザーズガイドをよく読んで，基本的な操作方法を習得することをお勧めします．

ユーザ各位が，このリファレンスマニュアルを熟読され，優れた機能を持つBASICソフトウェアを，数多く作り出されることを期待いたします．

このマニュアルをお読みになる前に必ずグリーティングカードを御覧ください．

このマニュアルの構成

本書ではN_{80SR}-BASICの各機能，命令などをわかりやすくするために，1章，2章，3章そして資料という構成になっています．

第1章　N_{80SR}-BASICの概要

N_{80SR}-BASICの特長や機能をコマンド・ステートメントなどの説明に先だって解説しています．その内容は，BASIC言語の定数，変数の意味，演算の決まりなど，BASIC言語を使う上での最低限の知識と，N_{80SR}-BASICに新しく追加された機能の紹介や，それを使う上での知識です．これらは2章の文法の説明を理解する上でどうしても必要な事項です．必ず読むようにしてください．

第2章　コマンド・ステートメント・関数・予約変数

N_{80SR}-BASICで使用することができるコマンド・ステートメント・関数・予約変数について解説します．

第3章　N_{80SR}-BASICの応用

ここでは第2章で説明したN_{80SR}-BASICの命令を使って，プログラミングの方法やプログラムのデバッグ方法，そして機械語プログラムの呼び方について説明します．

資　料

N_{80SR}-BASICの予約語やエラーメッセージ，FM音源とシンセサイザーICの解説などプログラミングに必要な資料で構成されています．

さらに，PC-8001MKⅡSRとPC-8000/8800シリーズとの関係やN_{80}-BASICとN-BASICの解説もここでまとめて取り扱っています．

このマニュアルで使われている記号

このマニュアルでは解説をわかりやすくするために，次の記号を使っています．

⏎	リターンキーの入力を表します．
□	スペースを表します．
注意：	特に注意が必要な事柄であることを表します．
XXH	数字にHをつけて16進数を表します．

N80SR-BASIC REFERENCE MANUAL

目　次

第１章　N80SR-BASICの概要

第2章　コマンド・ステートメント・関数・予約変数

第３章　N80SR-BASICの応用

資　料

機能別索引

1. 一般的な命令

一般コマンド

一般ステートメント

2. 入出力に関する命令・関数

入出力コマンド

フロッピィディスク・カセットテープ

入出力ステートメント

フロッピィディスク

カセットテープ

画面・キーボード・プリンタ

スピーカ

RS-232C通信ポート

入出力関数

フロッピィディスク

キーボード・プリンタ

RS-232C通信ポート

3．画面制御に関する命令・関数

画面制御ステートメント

画面制御関数

画面制御予約変数

4. グラフィックスに関する命令・関数

グラフィックステートメント

グラフィック関数

5. ミュージックに関する命令

ミュージックステートメント

6．算術演算に関する関数

数値関数

7．文字列操作に関する命令・関数

文字ステートメント

文字関数

文字予約変数

8．特殊な命令や関数

特殊コマンド

特殊ステートメント

特殊関数

特殊予約変数

第1章

N80SR-BASICの概要

1.1　N_{80SR}-BASICの主な特長

N_{80SR}-BASICは新しい機能として次のような特長を持っています.

1. N_{80SR}-BASICはN-BASICをもとにグラフィック機能，サウンド機能を中心に強化されたBASICです．一部の命令を除いてN-BASICと互換性がありますので，N-BASICからの変更が容易に行えます.
2. 48Kバイトのグラフィック専用RAMを持ち，次のような多種のグラフィックモードをサポートします.
    1. 640×200　8色カラー　1画面
    2. 640×200　白黒　3画面(合成表示可)
    3. 320×200　8色カラー　2画面(優先順位付合成表示可)
    4. 320×200　白黒　6画面(合成表示可)
3. 独立したテキスト画面を持ち，グラフィック画面と合成して表示することができます．このとき，テキスト画面とグラフィック画面の間に優先順位を付けることもできます.
4. グラフィックの色の指定はパレットを使って行うことができますので，多彩なカラーを表示することができます.
5. シンセサイザーIC(略称OPN…FM Operator Type-N)を内蔵しており，FM音源3声，SSG音源3声による6重奏が可能です.

    さらに，オーディオアンプ等を接続することによって，本格的な演奏を楽しむことができます.
6. 漢字ROMボード(別売)を接続すれば，日本語の文章を表示することができます.

1.2　動作モード

本体の電源を入れBASICを起動すると画面にメッセージが表示され，そのあと "Ok" という文字が表示されます．

この "Ok" が表示されている状態を「BASICがコマンドレベルにある.」といいます．BASICがコマンドレベルにあるときは，そのBASICのあらゆるコマンドをキーボードから入力することができます．BASICのコマンドは，ダイレクトモード，プログラムモードのいずれかで使用することができます．

1.2.1　ダイレクトモード

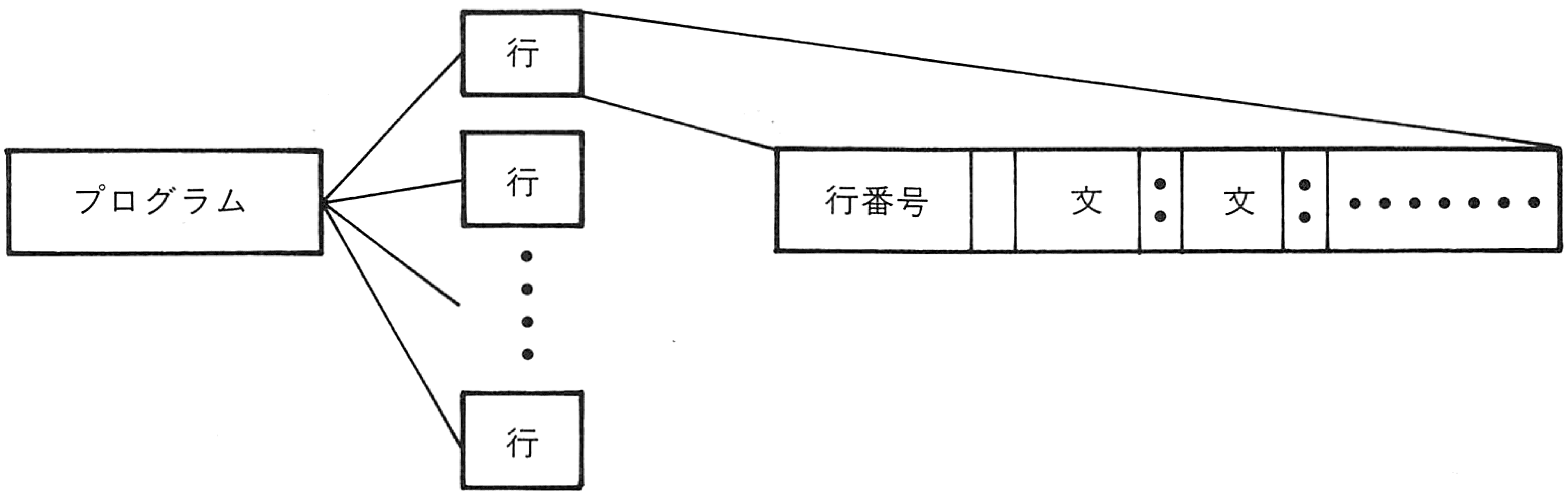
行番号を付けずにBASICの文法に沿った文をキーボードから入力し，[↵]キーを押すと，その文はすぐに実行されます．これをダイレクトモードの実行といいます．入力された文はメモリには残りません．

1.2.2　プログラムモード

行番号(0～65529までの整数)を付けて文を入力した場合，[↵]キーを押してもすぐには実行されず，メモリの中にプログラムとして行番号順に格納されます．この格納されたプログラムは，RUNコマンド，GOSUB文，GOTO文により実行させることができます．これをプログラムモードの実行といいます．なお，1つの行を入力後[↵]キーを押した場合，"Ok" の表示はされませんが，BASICはコマンドレベルにあります．

1.3　行の形式

BASICのプログラムは行の集まりによって構成されます．そして行は，行番号と1つまたは複数の文から成り立っています．



文はBASICが行う手続き(PRINT A, C＝SIN(B), GOTO 100など)を表している最小単位です．コロン(:)で区切ることにより，1つの行に2つ以上の文を入れることができます．これを複文(マルチステートメント)と言います．なお，1行(行番号を含む)の長さは255文字まで許されています．

1.4　行番号

BASICの各プログラム行は行番号で始まらなければなりません．行番号には0～65529までの整数を用います．行番号はプログラム行をメモリに格納する順序を示し，実行も行番号の若い方から行われます．また分岐や編集の目印としても使用されます．

行番号の代わりにピリオド(．)を使うことができる場合があります．

ピリオドは，エラーが発生した行，あるいは編集時において現在の行の行番号の値を持ち，LISTコマンド，AUTOコマンド，DELETEコマンド等で使うことができます．

例)

```
10 PRINT  "abc"
20 PLINT  "def"
30 PRINT  "ghi"

run
abc
Syntax error in 20
Ok
list .
20 PLINT  "def"
Ok
```

1.5 特殊記号の使い方

BASICでは，演算子(＋，－，＊，/など．1.10 式と演算参照)の他にも特別な意味を持つ記号があります．ここでその意味をまとめて説明しておきます．

1. ダブルクォーテーション(")

ダブルクォーテーションで囲まれた内容を文字列として扱います．文字をダブルクォーテーションで囲んで表示させた場合と，囲まずに表示させた場合の違いは次のようになります．

例)

```
10 ABC=123
20 PRINT "ABC" ←――文字列として扱う
30 PRINT ABC   ←――変数として扱う
40 END

run
ABC
 123
Ok
```

2. ピリオド(．)

ピリオドは，エラーが発生した行，または編集時において現在の行の行番号の値を持ちます．

例)

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT
40 END

list 40
40 END
Ok
delete
Ok
list
10 FOR I=1 TO 10
20 PRINT I
30 NEXT
Ok
```

3. ハイフン(－)

LISTコマンド，DELETEコマンドなどで行の範囲を指定するとき，何行から何行までと

いう場合に使います.

例)　　delete 100－200

4. コロン(:)

マルチステートメントの区切りとして使います. マルチステートメントとは, 1行(255文字以内)に複数の命令を書いたものです.

例)

```
10 A=B+C
20 PRINT A     ←→   10 A=B+C:PRINT A
```

5. コンマ(,)

PRINT文, INPUT文などパラメータが並ぶ場合, その区切りとして使います.

例)

```
input a,b,c
color 7,,,7
```

6. セミコロン(;)

PRINT文などの区切りとして使います. この場合, 前の表示のあとに続けて表示することができます.

例)

```
10 PRINT "A=",123
20 PRINT "B=";456

run
A=              123
B=  456
Ok
```

7. アポストロフィ(')

REM(リマーク)文の代用として使えます.

例)

```
rem test
' sample
```

8. クエスチョンマーク(?)

PRINT文の代用として使えます.

例)

```
? 2*2
 4
```

9. イコール(=)

"=" は次の2つの意味で用いられます.

① 代入文を表します.

例)　　pi=3.14159

② 関係演算子で等号を表します. 詳しくは1.10.2 関係演算を参照してください.

1.6 スクリーンエディタ

N_{80SR}-BASICは, ディスプレイを使用してプログラムの作成および修正が簡単に行えるように,「スクリーンエディタ」と呼ばれる画面編集機能を備えています. この機能により, 修正したい行を最初から改めて入力する必要がなくなり, プログラムの入力や訂正を短時間で行うことができます.

以下に出てくるキー操作の説明の中で[CTRL]+[H]などのような記述は, [CTRL]キーを押しながら[H]キーを押すことを示します.

1.6.1 入力中の修正およびエラーの訂正

プログラムを入力している途中で誤りに気付き, それを修正したい場合, あるいは実行した結果エラーが生じたプログラムを訂正したい場合には, 次のような方法を使うことができます.

(1) カーソルの直前の文字の消去

カーソルの前の文字を消す場合, [INS DEL]を押します. [CTRL] + [H]でも同じ働きをします.

(2) 文字の修正

押し間違えた文字を正しい文字に置き換えたい場合は, カーソル移動キー([↕] [⇆])と[SHIFT]キーを使用して, 間違えた文字のところへカーソルを移動し, 正しい文字を入力します. [↕]を押すとカーソルは上方向へ移動し, [SHIFT]+[↕]を押すと下方向へ移動します. また[⇆]を押すとカーソルは右方向へ移動し, [SHIFT]+[⇆]を押すと左方向へ移動します.

(3) 文字の消去

押し間違えた文字を消去したい場合は, カーソル移動キー([↕] [⇆])と[SHIFT]キーを使用して, カーソルを消したい文字の次の文字の所へ移動し, [INS DEL]を押します. [INS DEL]を1回押した場合は, カーソルの前の1文字が消され, カーソルから右の文字列が1文字分左へ移動します. (ただし, [INS DEL]キーを約1秒間以上押し続けた場合, 連続して文字が消去されます. 同様に, 他のキーを約1秒間以上押し続けても, 連続して文字が表示されます.)

(4) 文字の挿入

文字を挿入する場合は，挿入したい位置にカーソルを移動させ，挿入する文字の数だけ [SHIFT] ＋ [INS/DEL] を押してから文字を入力します．[SHIFT]＋[INS/DEL] は [CTRL]＋[R] でも代用できます．

スクリーンエディタの機能を用いてプログラムを訂正した場合，各行の訂正が終わる度に必ず[↵]キーを入力してください．

1.6.2 便利な編集機能

PC-8001MKⅡSRは，前節で説明した機能の他にも次のような編集機能を持っています．

(1) カーソルからその行の終わりまでの消去…… [CTRL]＋[E]

カーソルが点滅している文字からその行の終わりまでを消去する場合は [CTRL] ＋ [E]を入力します．この後で[↵]キーを入力すれば，メモリの内容も訂正されます．ですから，行番号のすぐ次の位置にカーソルを移動し，[CTRL]＋[E] を入力してから [↵]キーを入力すると，その行番号の文はプログラムから削除されます．(行番号だけを入力し，その後 [↵] キーを入力しても，その行をプログラムから削除することができます.)

(2) カーソルの移動…… [CTRL]＋[N]，[CTRL]＋[B]

[CTRL] ＋[N]を入力することによって，カーソルを次の項目の先頭に移動させることができます．また，[CTRL]＋[B]を入力すると，カーソルを1つ前の項目の先頭に移動させることができます．

ここでいう項目とは，"0"～"9"，"A"～"Z"，"a"～"z"，"。"(キャラクタコードA1H)～"°"(キャラクタコードDFH)で始まる文字列のことです．

(3) 行の分割

1行の内容を2行に分割したい場合は，区切りたい場所の最初の文字にカーソルを移動し，[CTRL]＋[J]を押します．カーソルのある文字以降の文が次の行に移動します．カーソルは次の行の先頭に移ります．行番号の文字数分 [SHIFT] ＋[INS/DEL] を押した後，行番号を入力し[↵]キーを押します．これにより，1行の内容を2行に分割することができます．

例)
(a) 10 INPUT A$▒PRINT A$
　　　　　　　　↑── カーソルを意味します．
(b) 10 INPUT A$
　　▒PRINT A$
(c) 10 INPUT A$
　　20 PRINT A$

(a)ではコロン(：)の所にカーソルを移動させ，[CTRL]＋[J]を押しています．これにより，コロン以降の文が次の行に移動します(b)．コロンを消去し，行番号を付けることに

よって，(c)のように２行に分割することができます．

その他のコントロールコードについては，資料7　コントロールコード表を参照してください．

1．7　定　　数

1．7．1　定数の種類

N_{80SR}-BASICの定数には以下のものがあります．

- 定数
  - 文字定数
  - 数値定数
    - 整数型
      - 10進形式
      - ８進形式
      - 16進形式
    - 実数型
      - 単精度型
      - 倍精度型

1．7．2　文字定数

文字定数とは，ダブルクォーテーション(")で囲まれた255文字以下の英数字，カナ文字，記号などの文字列のことです．また，何もない文字列のことをヌルストリングといいます．なお，(")を文字列の中に記述する場合はCHR$関数を用います．

例）

```
print "Good Morning"
Good Morning

print chr$(34)+"Hello"+chr$(34)
"Hello"
```

1．7．3　数値定数

数値定数は，整数型，実数型に分けられ，それらは正あるいは負の数，または０です．

負の数の前には必ずマイナス符号を付けなければなりませんが，正の数の前のプラス符号は省略することができます．

1．7．4　整数型

(1)　10進形式

－32768～＋32767までのすべての整数，または数の後に％(整数型を表す型宣言文字)を

付けたものです．ただし小数点を付けることはできません．

例）　32767

　　　−123

　　　32767% ←――整数を表す．

⑵　8進形式

頭に&O(&o)または&を伴った，0から7までの数字の並びで表します．扱うことができる範囲は &0 ~ &177777 までで，&0 ~ &77777 は正の数を，&100000 ~ &177777 は負の数を表します．

例）　&12345（$5349_{(10)}$）　　$_{(10)}$は10進数を表します．

　　　&o7777（$4095_{(10)}$）

　　　&123456（$-22738_{(10)}$）

⑶　16進形式

頭に&H(&h)を伴った，0からFまでの並びで表します．扱える範囲は&H0～&HFFFFまでで，&H0～&H7FFFまでは正の数を，&H8000から&HFFFFまでは負の数を表します．

例）　&h100（$256_{(10)}$）

　　　&h7fff（$32767_{(10)}$）

　　　&h9000（$-28672_{(10)}$）

注意：8進形式または16進形式で入力された数値は，PRINT文，LPRINT文では10進形式で出力されます．

例）　　10進数，8進数，16進数の関係

10進数	8進数	16進数
0	&0	&H0
1	&1	&H1
2	&2	&H2
3	&3	&H3
4	&4	&H4
5	&5	&H5
6	&6	&H6
7	&7	&H7
8	&10	&H8
9	&11	&H9
10	&12	&HA
11	&13	&HB
12	&14	&HC
13	&15	&HD
14	&16	&HE
15	&17	&HF
16	&20	&H10
17	&21	&H11
18	&22	&H12
19	&23	&H13
:	:	:
60	&74	&H3C
61	&75	&H3D
62	&76	&H3E
63	&77	&H3F
64	&100	&H40
65	&101	&H41
66	&102	&H42
67	&103	&H43
68	&104	&H44
69	&105	&H45
:	:	:
250	&372	&HFA
251	&373	&HFB
252	&374	&HFC
253	&375	&HFD
254	&376	&HFE
255	&377	&HFF
256	&400	&H100
257	&401	&H101
258	&402	&H102
259	&403	&H103
:	:	:

1．7．5　実数型

実数型は，単精度型と倍精度型に分けられます．

1　単精度型

有効桁7桁の精度で格納されます．出力のときは7桁目が四捨五入され，6桁以下で表示さ

れます．

扱える数値の範囲は次のとおりです．

−1.70141E＋38～−2.93874E−39，

0，

2.93874E−39～1.70141E＋38

(1)　7桁以下の実数

(2)　E（大文字に限る）を使った指数形式（このEは10のべき乗を意味します）

(3)　最後に！（単精度実数型を表す型宣言文字）を伴った数

例）　1.23

−7.09E−06＝-7.09×10^{-6}＝−0.00000709

3525.68

3.14!

2　倍精度型

有効桁16桁の精度で格納され，16桁以下で表示されます．

扱える数値の範囲は次のとおりです．

−1.701411834604692D＋38～−2.938735877055719D−39，

0，

2.938735877055719D−39～1.701411834604692D＋38

(1)　8桁以上の数

(2)　D（大文字に限る）を使った指数形式（このDは10のべき乗を意味します）

(3)　最後に#（倍精度実数型を表す型宣言文字）を伴った数

例）　1234567890

0.3141592653D＋01＝0.3141592653×10^{1}＝3.141592653

56789.0#

8657036.1543976

1.8 変　　数

1.8.1 変数とは？

変数とは，数値や文字列を記憶させておく場所で，英数字からなる名前(たとえばCT，XPなど)が付いています．この名前を変数名と呼びます．

この変数を使うことによって，ある値を記憶させておいたり，取り出したり，あるいは参照したりすることができます．

例)
```
pi=3.14159    ←──記憶させる
print pi*2    ←──取り出す
```

定数と同じように変数にも数値型(数値変数)と文字型(文字変数)があります．数値変数は数値を記憶させておくもので，文字列を入れることはできません．逆に文字変数に数値を入れることもできません．数値変数は値が割り当てられる前に参照されたら0が代入され，文字変数はヌルストリング(空の文字列)が代入されます．

次の例は悪い例です．

例)
```
pi="Bad example"    ←──数値変数に文字列を代入しています．
a$=3.14159          ←──文字変数に数値を代入しています．
```

1.8.2 変数名と型宣言文字

N_{80SR}-BASICにおいて，変数名は何文字あってもかまいません．ただしN_{80SR}-BASICでは最初の2文字までの判別しかしません．また最初の文字は英字でなければなりませんが，残りの文字は英字または数字のどれでもかまいません．ただし変数名は資料3　N_{80SR}-BASICの予約語に示す予約語を含んではいけません．

例)
```
10 NEC=123
20 PRINT NEC
30 PRINT NED
40 PRINT NFD
50 END

run
 123
 123
 0
Ok
```

同じ変数名でも型が違えば区別されます．変数の型は型宣言文字によって決まり，型宣言文字は変数名の最後に付けて，その変数の型を表します．型宣言文字を省略すると，"!"が付いているとみなされます(単精度実数型変数となる)．

型宣言文字
- % 整数型
- ! 単精度実数型
- # 倍精度実数型
- $ 文字型

A
A#
A%
A$

これらは区別されますが，AとA!は同じで単精度実数型を意味します．

次の例は，実数を単精度型の数値変数に代入(10行)したにもかかわらず，20行で整数型の数値変数を指定したため0が表示された例です．

例)
```
10 AB=0.123456789
20 PRINT AB%

run
 0
Ok
```

このように，違った型の数値変数に代入されたものを参照する場合，変数名の指定には注意が必要です．

変数の型を宣言するのにもう一つ便利な方法があります．それはDEFINT，DEFSNG，DEFDBL，DEFSTRの型宣言文をプログラム中で用いる方法です．これについては2章で詳しく説明します．

1.8.3 配列変数

配列とは同じ性格のデータが複数個集まったものであり，その配列を記憶させておく場所を配列変数といいます．配列変数はDIM文(2章参照)で宣言し，要素(データ)は添字によって順序付けられます．次元は"添字の最大値"の個数によって決まり，1次元から多次元まで指定できます．

例)　　1次元配列　　　　2次元配列

dim a(5)

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

要素の数＝6

dim b$(2,3)

B$(0, 0)	B$(0, 1)	B$(0, 2)	B$(0, 3)
B$(1, 0)	B$(1, 1)	B$(1, 2)	B$(1, 3)
B$(2, 0)	B$(2, 1)	B$(2, 2)	B$(2, 3)

要素の数3×4＝12

各添字は0から始まりますから，実際の要素の数は添字の最大値＋1になります．添字の最大値は，整数形式で0〜32767までの範囲を指定します．ただし，メモリの空き領域の大きさによって添字の最大値および次元は制限されます．

また，添字の最大値が10以下のときは，DIM文による宣言を省略することができます．この場合，添字の最大値は10とみなされます．

要素の参照は，変数名と各次元の添字を指定することによって行います．

例)

```
10 DIM D(15)
20 FOR I=0 TO 15
30 D(I)=I          ←――要素を配列変数に格納
40 NEXT
50 PRINT D(12)     ←――添字が12の場合の要素を参照
60 END
```

1.8.4　予約変数

N80SR-BASICには，BASIC自身が専用に使う変数があります．これらの変数は，ユーザによって一般の変数として使うことはできません．(詳しい使い方は第2章を参照してください．)

TIME$　現在の時分秒をHH: MM: SSの形で持っています．また，時分秒を代入することもできます．

DATE$　現在の年月日をYY/MM/DDの形で持っています．また，年月日を代入することもできます．

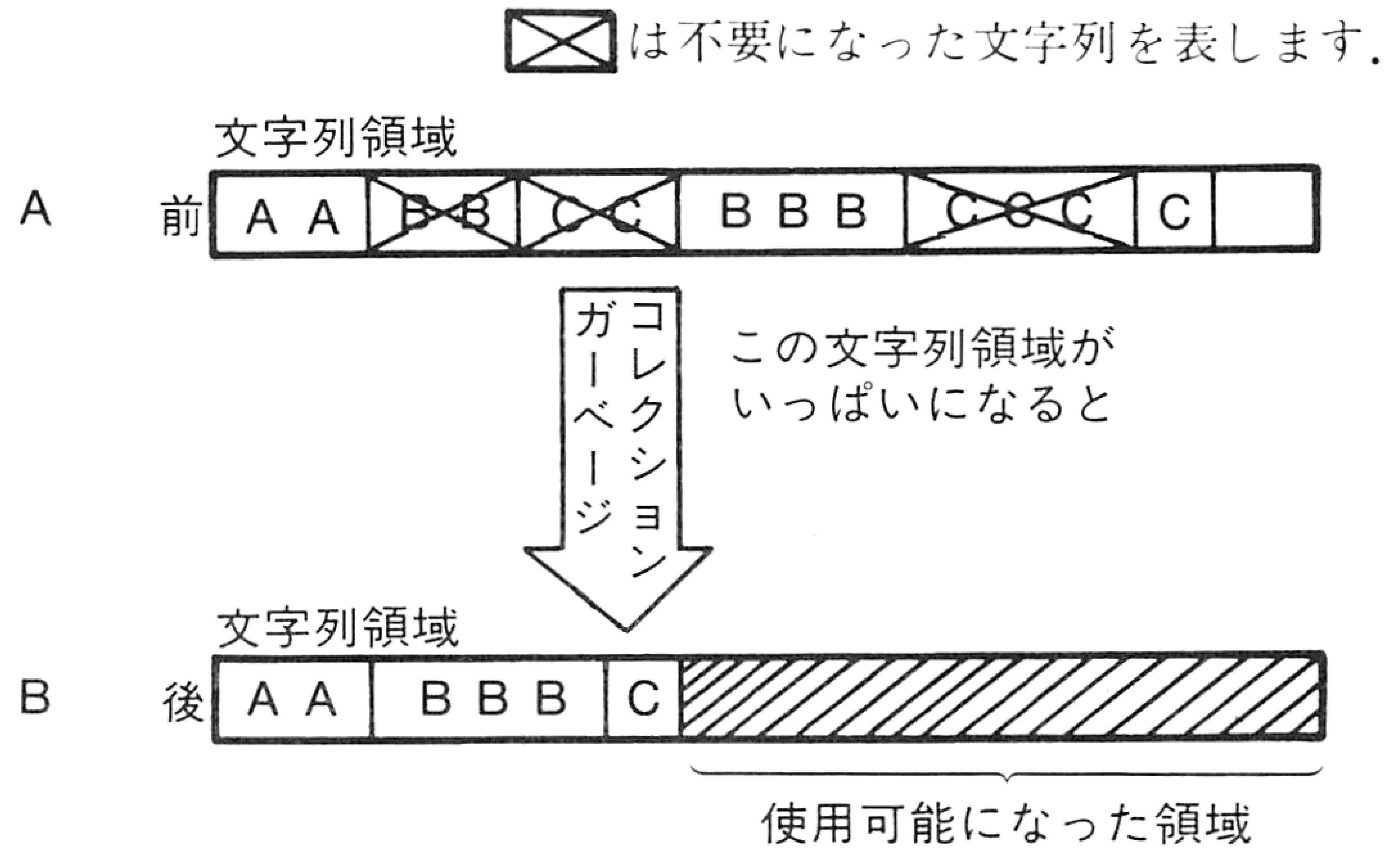
ERL　エラーが生じたとき，エラーが発生した行番号を持っています．代入するこ

とはできません．

ERR　エラーが発生したとき，生じたエラーのエラーコードを持っています．代入することはできません．

1.8.5　ガーベージコレクション

一般的に，文字列領域内には文字列が順番に存在するものではなく，あちこちに存在していて，その間には不要になった文字列があります(下図A)．そして，次々に文字変数に文字列が与えられ文字列領域がいっぱいになると，現在使われている文字列の領域と不要になった文字列の領域の2つに分けて使用可能な領域を確保します(下図B)．このことをガーベージコレクションと言います．



このガーベージコレクションは，プログラムを実行しているときなど自動的に行われ，場合によっては処理に時間がかかることがあります．文字列を多く扱うプログラムを実行中にPC-8001MKⅡSRがストップした状態になるのは，この処理を行っているためです．なお，ガーベージコレクションを行っている最中は，キーボードからの入力はききません．また，画面表示も停止したままとなります．

1.9　型変換

N_{80SR}-BASICの数値データは，必要に応じてその型から他の型に自動的に変換されます．

(1)　ある型の数値データが違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます．

例）

```
10 AB%=1.234   ←──実数を整数に変換
20 PRINT AB%

run
 1
```

(2)　精度の違う数値間の演算の場合，精度の高い方に変換されて演算が行われます．たとえば，10#/3 の場合は10#/3#として演算が行われます．

例）

```
print 10/3,10/3#,10#/3,10#/3#
 3.33333         3.333333333333333
 3.333333333333333
 3.333333333333333
```

(3)　論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます．

例）

```
print 12.34,not 12.34
 12.34         -13
```

NOTは否定を意味します．詳しい説明は，1.10.3　論理演算を参照してください．

(4)　実数が整数に変換される場合は，小数点以下は切り捨てられます．
このとき，整数型で扱える範囲を超えた場合はエラーが起こります．

例）

```
10 A%=34.4               10 A#=1.234E+07
20 B%=34.5               20 B%=A#
30 PRINT A%,B%           30 PRINT B%,A#

run                      run
 34          34          Overflow in 20
```

(5)　倍精度変数が単精度変数に代入されたときは，変数の値は有効数字７桁に四捨五入されたものとなります．単精度変数の精度は７桁であり，もとの倍精度の数値との誤差の絶対値は6.3×10^{-8}以下となります．

例）

```
10 A#=1.23456789#
```

```
20 B!=A#
30 PRINT A#,B!

run
 1.23456789          1.23457
```

1.10 式と演算

式とは定数や変数を演算子で結合させたものです．また，式の演算結果は1つの数値あるいは1つの文字列になりますから，単なる文字や数値，あるいは変数だけのものも式になります．式は単独で使用されることはなく，PRINT文や代入文などで引用されます．

例）
```
"BASIC"
3.14
10+3/5
a+b/c-d
tan(do)
```

BASICの演算は次の5つに分類されます．

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数
5. 文字列演算

1.10.1 算術演算

算術演算子には次のようなものがあります．

実行順序	演算子	演　算	例
1	^	指数演算	X^Y
2	−	負　号	−X
3	*，/	乗算，実数の除算	X*Y，X/Y
4	￥	整数の除算	X￥Y
5	MOD	整数の剰余	X MOD Y
6	+，−	加算，減算	X+Y，X−Y

演算の実行順序を変更する場合，カッコを使用します．カッコ中の演算子は他の演算子よりも先に実行されます．カッコ内においては上記の実行順序に従っています．

次に実行例を示します．

	代数表記	BASICの表記
1)	$2X+Y$	2＊X+Y
2)	$\frac{X}{Y}+2$	X/Y+2
3)	$\frac{X+Y}{2}$	(X+Y)/2
4)	X^2+2X+1	X ^ 2+2＊X+1
5)	X^{Y^2}	X ^ (Y ^ 2)
6)	$(X^Y)^2$	X ^ Y ^ 2
7)	$Y(-X)$	Y＊(−X)

(1) 整数の除算と剰余の計算

整数の除算は¥によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が切り捨てられます．商も小数点以下が切り捨てられた整数となります．

例）　10¥3———→3　　(10/3=3・・・余り1)

　　　23.75¥5———→4　(23/5=4・・・余り3)

剰余の計算はMODによって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が切り捨てられます．結果は整数の割り算の余りです．

例）　13.3 MOD 4———→1　　(13/4=3・・・余り1)

　　　25.68 MOD 6.99———→1　(25/6=4・・・余り1)

注意：¥，MODで扱うことができる数値は，小数点以下を切り捨てた値が−32768〜32767の範囲にある整数または実数です．この範囲外の数値を指定したときは "Overflow" エラーになります．

(2) 0での除算

式の実行時に0での除算が行われた場合は，エラーメッセージを出力し，コマンドレベルに戻ります．

また，べき乗の実行時に，0に対して負のべき乗を行った場合も同様になります．

例）

```
print 2/0
Division by zero

print 0 ^ −1
Division by zero
```

(3) 桁あふれ(オーバーフロー)

代入や演算の結果がその変数の型内で表現することのできる範囲を超えた場合，桁あふれが発生します.

桁あふれが起こった場合，BASICは "Overflow" エラーを出力します.

例）
```
print 300 ^ 300
Overflow
```

(4) 0のべき乗，0乗

0のべき乗は次のとおりに定義されています。

0 ^ (正の数)———→ 0

0 ^ 0 ———→ 1

0 ^ (負の数) ———→ "Division by zero" エラー

また，0乗を行った場合は結果としてすべて1が返されます.

1.10.2 関係演算

関係演算子は2つの数値を比較するときに用います. 結果は，真(−1)，偽(0)で得られ，条件判定文などプログラムの流れを変えるのに用いられます(第2章 IF文参照).

関係演算子	内　　容	例
＝	等しい	X＝Y
＜＞，＞＜	等しくない	X＜＞Y，X＞＜Y
＜	小さい	X＜Y
＞	大きい	X＞Y
＜＝，＝＜	小さいか等しい	X＜＝Y，X＝＜Y
＞＝，＝＞	大きいか等しい	X＞＝Y，X＝＞Y

注意：＝は代入文にも使います.

IF文の中での使い方の例を次に示します.

例）
```
if x=0 then 1000
if a+b<>0 then x=x+1:y=y+1
```

1.10.3 論理演算

論理演算子は，複数の条件を調べたり，ビット操作や論理演算を行ったりするときに用いま

す．論理演算はビットごとに０または１を結果として与えます．各論理演算の内容を次に示します．(この記述は数値としての１，０ではなく，各ビットの１，０を示しています.)

NOT＝not(否定)

X	NOT X
1	0
0	1

AND＝and(論理積)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR＝inclusive or(論理和)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR＝exclusive or(排他的論理和)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP＝implication(包含)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV＝equivalence(同値)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

論理演算も関係演算子のようにプログラムの流れを変えるのに用いられます．この場合，論理演算子は２つ以上の関係演算子と結ぶことができます．

例） (1) if x＜0 or 99＜x then 1000

(2) if 0＜x and x＜100 then x＝0

(3) if not (a＝0) then 20

(1) xが負，または99より大きければ行番号1000へ飛ぶ．

(2) xが正で，かつ100より小さければxに0を代入する．

(3) aが0でなければ行番号20へ飛ぶ．

注意：論理演算で扱うことができる数値は－32768から32767の整数で，この範囲外を指定したときには "Overflow" エラーになります．

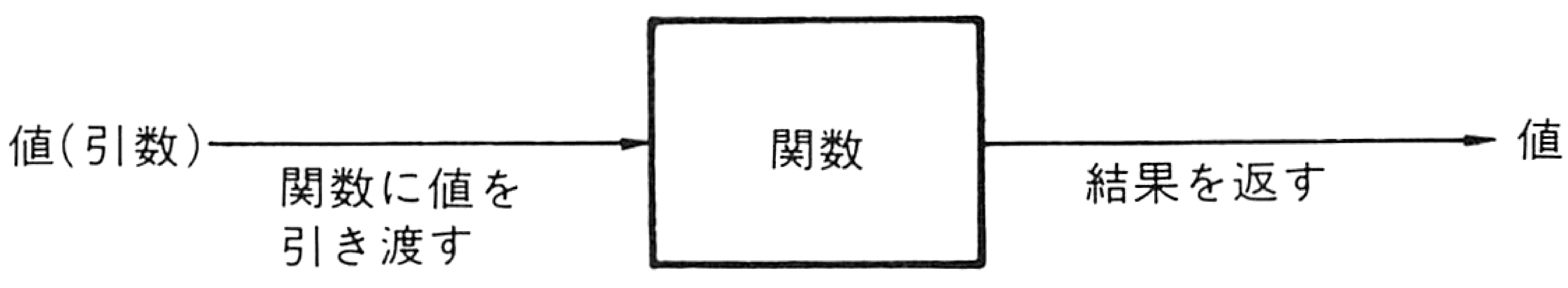
以下に論理演算子がどのように働くかを例をあげて示します．例の中で$(0011)_2$とあるのは（ ）内の数字が2進数で表記されていることを表します．

例）

NOT 2→－3	2＝$(0000000000000010)_2$
	したがってNOT 2＝$(1111111111111101)_2$＝－3
－1 AND 3→3	－1＝$(1111111111111111)_2$，3＝$(0011)_2$
	したがって－1 AND 3＝$(0000000000000011)_2$＝3
4 OR 3→7	4＝$(0100)_2$，3＝$(0011)_2$
	したがって4 OR 3＝$(0111)_2$＝7
－2 XOR 4→－6	－2＝$(1111111111111110)_2$，4＝$(0100)_2$
	したがって－2 XOR 4＝$(1111111111111010)_2$＝－6
－2 IMP －3→－3	－2＝$(1111111111111110)_2$，－3＝$(1111111111111101)_2$
	したがって－2 IMP －3＝$(1111111111111101)_2$＝－3
5 EQV －4→6	5＝$(0101)_2$，－4＝$(1111111111111100)_2$
	したがって5 EQV －4＝$(0000000000000110)_2$＝6

1.10.4 関　数

ある値(引数)を引き渡して，決まった演算処理を行った結果を値として返してくるものが関数です．図で示すと次のようになります．



関数は単独で使われることはなく，PRINT文や代入文などで引用されます．

BASICは“組み込み関数”としてSIN(正弦)，SQR(平方根)などの数値関数やCHR$，LEFT$などの文字関数を本体内に持っています．

また，BASICは“ユーザ定義関数”としてユーザが自由に定義できる関数機能を持っています．これは2章のDEF FN文で説明します．

引数に整数しかとらないものは，小数部分を切り捨ててから演算を行います．

次に関数の使い方の例を示します．

例)
```
a=sin(3.14)+cos(3.14)
print sqr(2)
```

1.11 文字列の演算

1.11.1 文字列の連結

文字列は演算子"+"によって連結することができます．

例)
```
10 A$="NEC":B$="Personal":C$="Computer"
20 D$=A$+" "+B$+" "+C$
30 PRINT D$

run
NEC Personal Computer
```

1.11.2 文字列の比較

文字も数値の比較に用いられるものと全く同じ関係演算子を用いて比較することができます．

=，<，>，< >，><，<=，=<，>=，=>

文字列の場合，それぞれの文字列の最初から1文字ずつ文字の比較を行います．もし相互に全く同じ文字列の場合は，その2つの文字列は等しくなりますが，1箇所でも違った場合は，その文字のキャラクタコード(資料8　キャラクタコード参照)の大きい方の文字列が大きくなります．文字列の片方が短くて比較が途中で終わった場合は，短い文字列の方が小さくなります．

文字列の比較においては空白なども意味を持ちますから注意してください．

例)
```
"AA"<"AB"
"BASIC"="BASIC"
```

"X&" > "X#"

"PEN□" > "PEN"

"cm" > "CM"

"DESK" < "DESKS"

このように文字列の比較は，文字列の内容を調べたり，文字をアルファベット順に並べたり(ソート)することに使うことができます．

1.12 演算の優先順位

演算は次の順位によって行われます．

1. カッコで囲まれたもの
2. 関数
3. 指数(べき乗) ^
4. 負号(－)
5. ＊，/
6. ¥
7. MOD
8. ＋，－
9. 関係演算子(<，>，＝など)
10. NOT
11. AND
12. OR
13. XOR
14. IMP
15. EQV

式の中の演算は上記の演算順位で実行されます．同じ優先順位の演算子のときは，左側から先に実行されます．

1.13　数値演算における誤差

通常の演算方式でプログラムを組んだところ，どうしても正しい答えが表示されないときに参照してください．

1.13.1　誤　　差

コンピュータでの計算が紙の上での計算と全く同じと考えている方が多くいらっしゃると思いますが，次のように必ずしも同じであるとはいえないことがあります．

例）　(1)　実数型変数のプログラム

```
100 C=0:A=0
110 PRINT "C=";C,"A=";A
120 IF A=1 THEN 150
130 C=C+1:A=A+0.1
140 GOTO 110
150 END
```

このプログラムは，カウントCと変数Aを0にしてから，変数Aが1となったところで終わるものです．カウントCは1ずつ，変数Aは0.1ずつ増えていきます．このプログラムを実行してみましょう．

```
run
C=0          A=0
C=1          A=.1
C=2          A=.2
C=3          A=.3
C=4          A=.4
C=5          A=.5
C=6          A=.6
C=7          A=.7
C=8          A=.8
C=9          A=.9
C=10         A=1
C=11         A=1.1
C=12         A=1.2
C=13         A=1.3
```

不思議なことに，A＝1となってもプログラムは止まりません．

例）　(2) 関数のプログラム

```
100 S=15
110 FOR Y=0 TO S
120 A=2^Y
130 PRINT "2^";Y;"=";A
140 NEXT Y
150 END
```

このプログラムは2^0から2^{15}までを計算して表示するものです．実行させてみましょう．

```
run
2^0=1
2^1=2
2^2=4
2^3=8
2^4=16
2^5=32
2^6=64
2^7=128
2^8=256
2^9=512
2^10=1024
2^11=2048
2^12=4096
2^13=8192.01
2^14=16384
2^15=32768
```

ここで，2^13に注目してください．8192.01というように，小数点以下2桁まで表示されています．

例）　(3) 数値表示プログラム

```
100 A#=0.0075
```

```
110 PRINT A#
120 END
```

このプログラムを実行してみましょう.

```
run
 7.500000298023224D-03
```

例の(1)のプログラムでは，A＝1となっているのにプログラムが止まらず，(2)と(3)のプログラムでは予想に反した結果が表示されます.

その理由を簡単に説明すると次のようになります.

(1)のプログラムでは，A＝1と表示されたときに，実はちょうど1になっていないからです．この現象は，コンピュータにおける数値の内部表現形式が原因となって起こります．後で詳しく説明します.

(2)のプログラムで小数点が現れたのは，関数の値を近似計算式で求めているためです．そして，他の関数(SIN, COS, TAN, etc.)についても同じように近似計算を行っているための誤差が含まれて出力されます.

(3)の場合は，データを内部表現形式に変換する場合に生まれる誤差の影響です.

1.13.2 対　策

それでは，どうすれば誤差の影響の少ないプログラムになるでしょうか．主な対策をあげてみましょう.

a. 実数型変数を比較する場合は等号(＝)ではなく，不等号(>，<)を使用します.

b. FOR～NEXTループの変数には整数型変数を使用します.

c. 結果を整数値として表示する場合はINT関数やFIX関数を使用します．このとき注意しなければならないのは，理論上，結果が

　　a＝1

となるものが

　　a＝0.9999……

となっている場合です．このとき，このままでINT関数を使用すると，

```
print int(a)
 0
```

となり，結果が0になってしまうときがあります．これは，INT関数が与えられた引数の整数部分だけを取り出すからです．このような場合は，INT関数に代入する前に0.1を加えておくという手段が有効です.

d. 結果を画面やプリンタに出力する場合はPRINT USING文を利用します.

例(1)の場合には，aに従い，行番号120の文を次のように変更します．これで小数点以下４桁の精度を必要とする場合の比較を行うことができます．

```
120 IF 0.9999<A AND A<1.0001 THEN 150
```

単精度の数値は有効桁６桁，倍精度は有効桁16桁ですので，このことを考慮して精度を決定してください．

1.13.3　数値の内部表現形式

一般にコンピュータ内部での数値の演算方式には次の２通りの方式があります．

２進演算方式……入力された数値を計算機内部で２進数に変換した後，いろいろな演算を行う．

例)　　入力値　$123_{(10)}$

　　　　内部値　$1111011_{(2)}$

10進演算方式……入力された数値を10進数のまま計算機内部に記憶し，演算を行う．

例)　　入力値　$123_{(10)}$

　　　　内部値　$\underline{00000001}_{(2)}$ $\underline{00000010}_{(2)}$ $\underline{00000011}_{(2)}$

　　　　　　　　→ $123_{(10)}$ ←

注意：$_{(2)}$ は２進数を表し，$_{(10)}$ は10進数であることを表しています．

これら２つの演算方式の大きな違いは，数値を２進数に変換するところです．つまり，10進数の形で残すか，２進数の形に変換してしまうかによって，再び10進数として表示する場合に誤差が生まれるかどうかが決まってきます．

たとえば，0.83を２進演算用に変換すると次のようになります．

$0.83=1/2^1+1/2^2+1/2^4+1/2^6+1/2^{10}+1/2^{11}+1/2^{12}+\cdots\cdots$

したがって，計算機内部では，

1101010001111……

という形になります．しかし，計算機のメモリには限度があり，通常は32ビットとか64ビットというように制限されています．したがって，再び10進数に変換すると，前とは異なった値になります．

ですから，２進演算方式を採用しているPC-8001MKⅡSRでは多少の誤差が現れますが，これは特に異常ということではありません．

ではなぜ多くのパーソナルコンピュータが２進演算を採用しているのでしょうか．主な理由は次の２つです．

1. 演算を早く行う．
2. 大きな数値でも少ないバイト数(メモリ容量)で表現できる．

例）　　255$_{(10)}$

10進演算

00000010 00000101 00000101

（3バイト）

2進演算

11111111

（1バイト）

1.14　画面モード

テキスト画面，グラフィック画面の各画面モードについて解説します．なお，N$_{80SR}$-BASICはテキスト画面とグラフィック画面を独立して持っていますので，テキスト画面を変化させてもグラフィック画面が影響を受けることはありません．また，その逆の場合も影響を受けません．

1.14.1　テキスト画面

キー入力された文字などを表示する画面をテキスト画面と呼びます．

テキスト画面の表示行数は20行と25行であり，また1行に表示できる桁数は80桁と40桁が許されています．これらの設定はWIDTH文を使って行います．（ただし，カラーモードを指定した場合，1行の桁数は79桁または39桁になります．）

テキスト画面

表示文字数（横×縦）
40×20
80×20
40×25
80×25

さらに，N$_{80SR}$-BASICではどの表示文字数のときでも白黒モードとカラーモードを選ぶことができます．

カラーモードの場合，8色のカラーをキャラクタ（文字）単位で設定することができます．また，白黒モードの場合，リバース，ブリンク，シークレットの3つの機能を組み合わせることにより，8つの機能をキャラクタ単位で設定することができます．ただし，これらの色や機能

を1行中で19回以上指定しても，19回目以降は無視されます．

モードや色，機能の設定の仕方は，2章のCOLOR文，CONSOLE文を参照してください．

1.14.2 グラフィック画面

グラフィック画面とは，図形を描いたり，漢字などのグラフィックパターンを表示したりする画面のことです．

N_{80SR}-BASICで使用するグラフィック画面には，次の表に示すような4種類の画面モードがあります．

画面モード

画面モード	解像度	ページ数
カラーモードⅠ	640×200	1
白黒モードⅠ	640×200	3
カラーモードⅡ	320×200	2
白黒モードⅡ	320×200	6

解像度とは，その画面がいくつのドット(点)で構成されているかを示します．

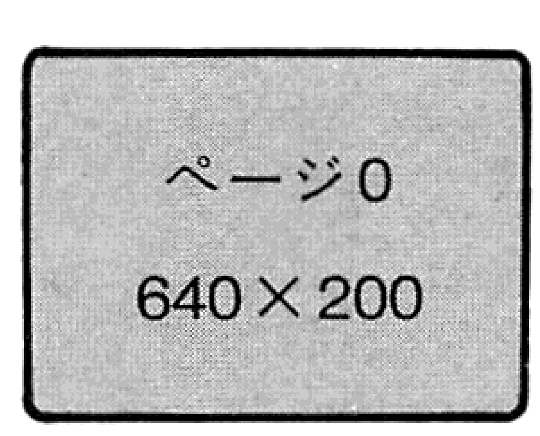
たとえば640×200とあるのは，1画面が横640個，縦200個の点の集まりから成り立っていることを表します．

ページ数とは，同時に何枚のグラフィック画面を持っているかを示すものです．

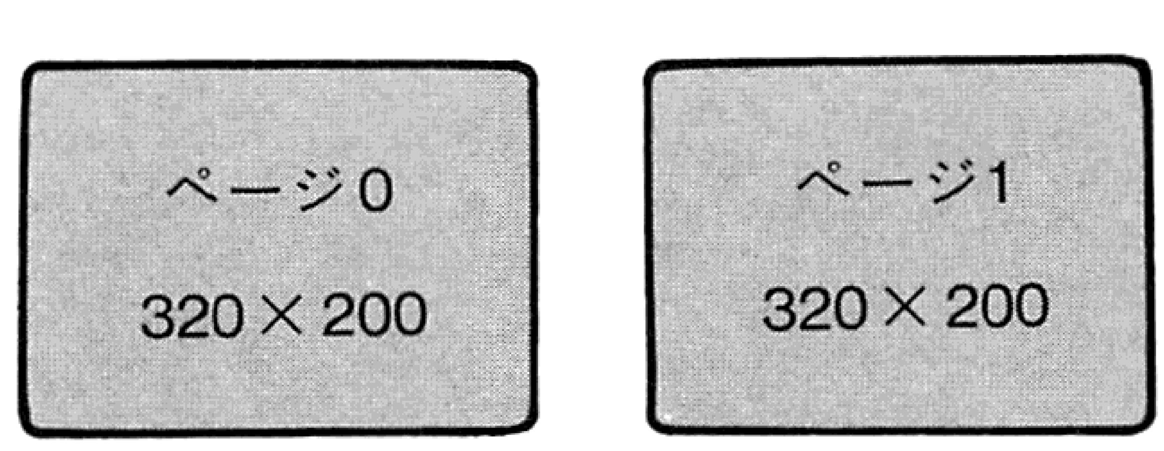
画面モードの切り替え方法については，2章のSCREEN文を参照してください．

(1) カラーモード

各ドットごとに任意の色(8色のうちから1色)が指定できる画面モードです．解像度，ページ数の違いにより，カラーモードⅠとカラーモードⅡがあります．(両方のモードを合わせてカラーモードと呼びます．)



カラーモードⅠ



カラーモードⅡ

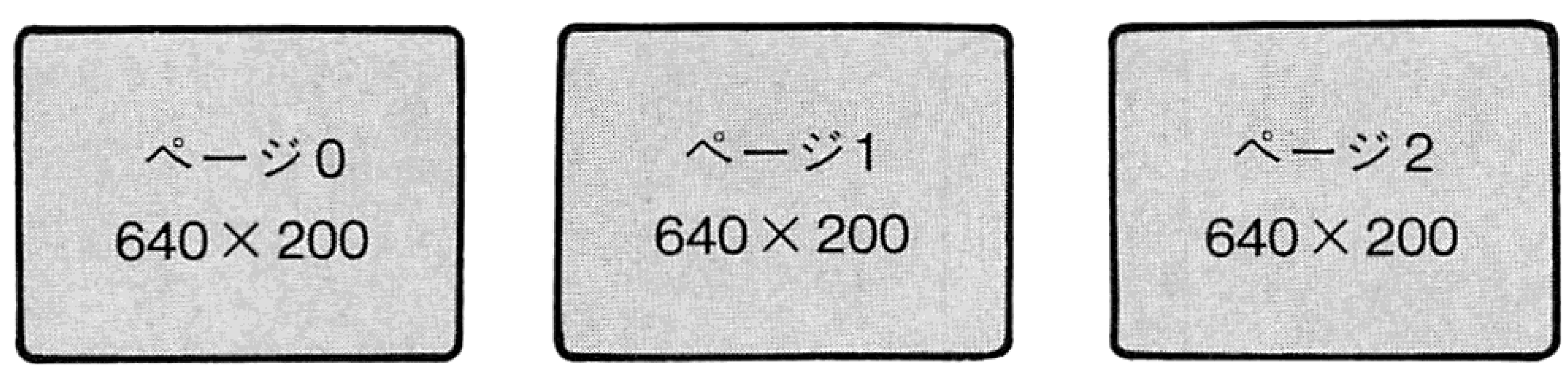
カラーモードⅡは2ページから成り，それぞれを独立に表示したり，合成して表示したりすることができます.

(2) 白黒モード

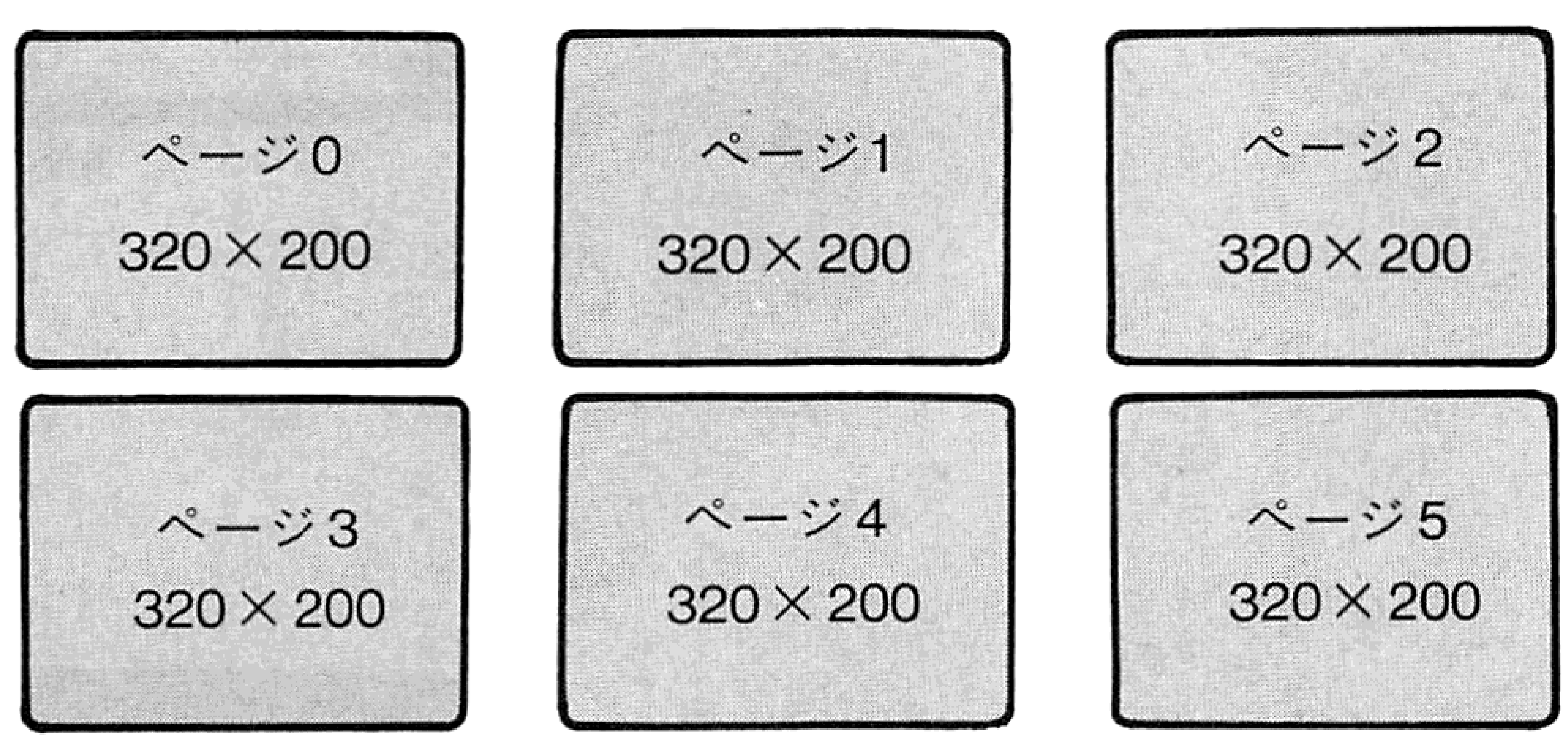
ドットがあるかないかで表示を行う画面モードです.

各ドットごとに色を付けることはできませんが，カラーモードに比べて3倍の数のページを持ちます.

解像度，ページ数の違いにより，白黒モードⅠ，白黒モードⅡがあります.（両方のモードを合わせて，白黒モードと呼びます.）



白黒モードⅠ



白黒モードⅡ

白黒モードでは複数のページを持ちますので，各ページを独立に表示させたり，任意のページを合成して表示させたりすることができます.

また白黒モードでは，各ドットに色を付けることはできませんが，テキスト画面のアトリビュート（文字の色や，リバースなどの機能）を使うことによりキャラクタ単位で色を付けることができます.

1.14.3 画面合成

PC-8001MKⅡSRは，テキスト画面とグラフィック画面を独立に持っており，これらは合成されて表示されます.

画面合成の方法は，カラーモードと白黒モードで異なります.

(1) カラーモードでの画面合成

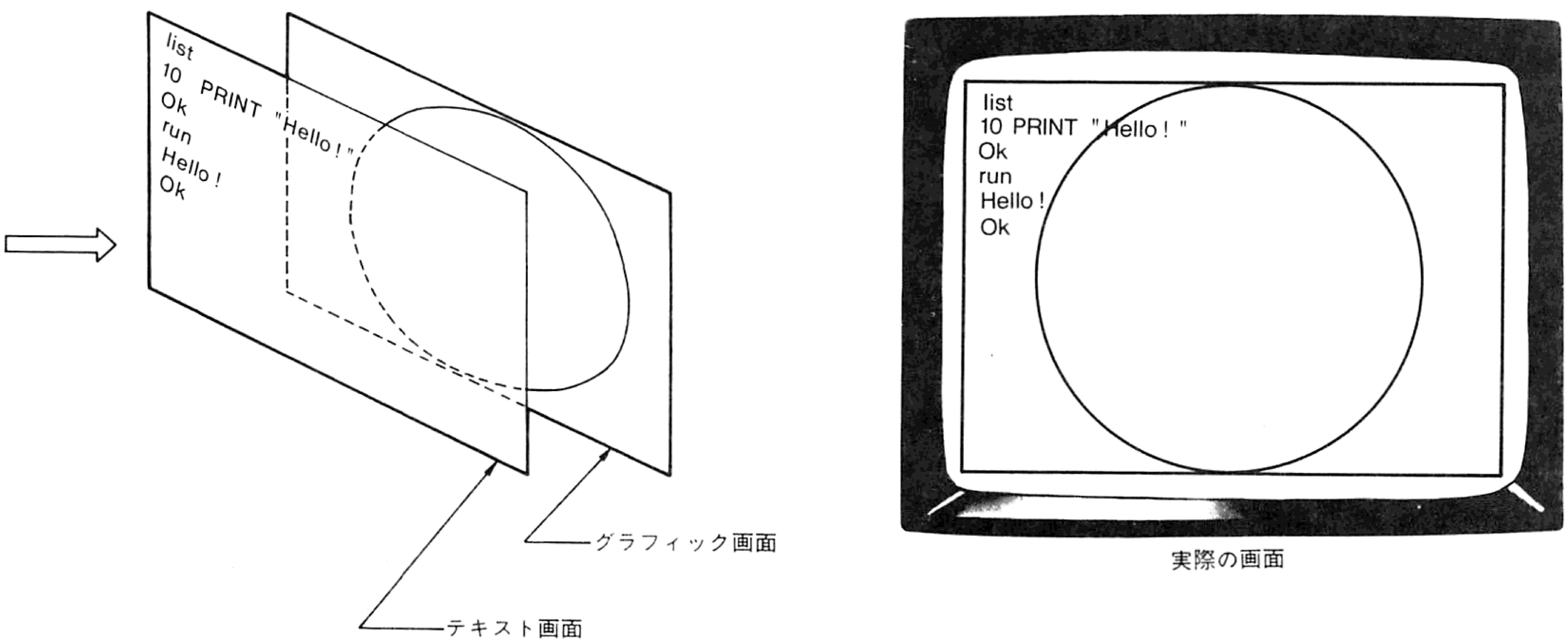
カラーモードでは，テキスト画面とグラフィック画面の間に優先順位が付いており，そ

の優先順位に従って合成されたものが表示されます．

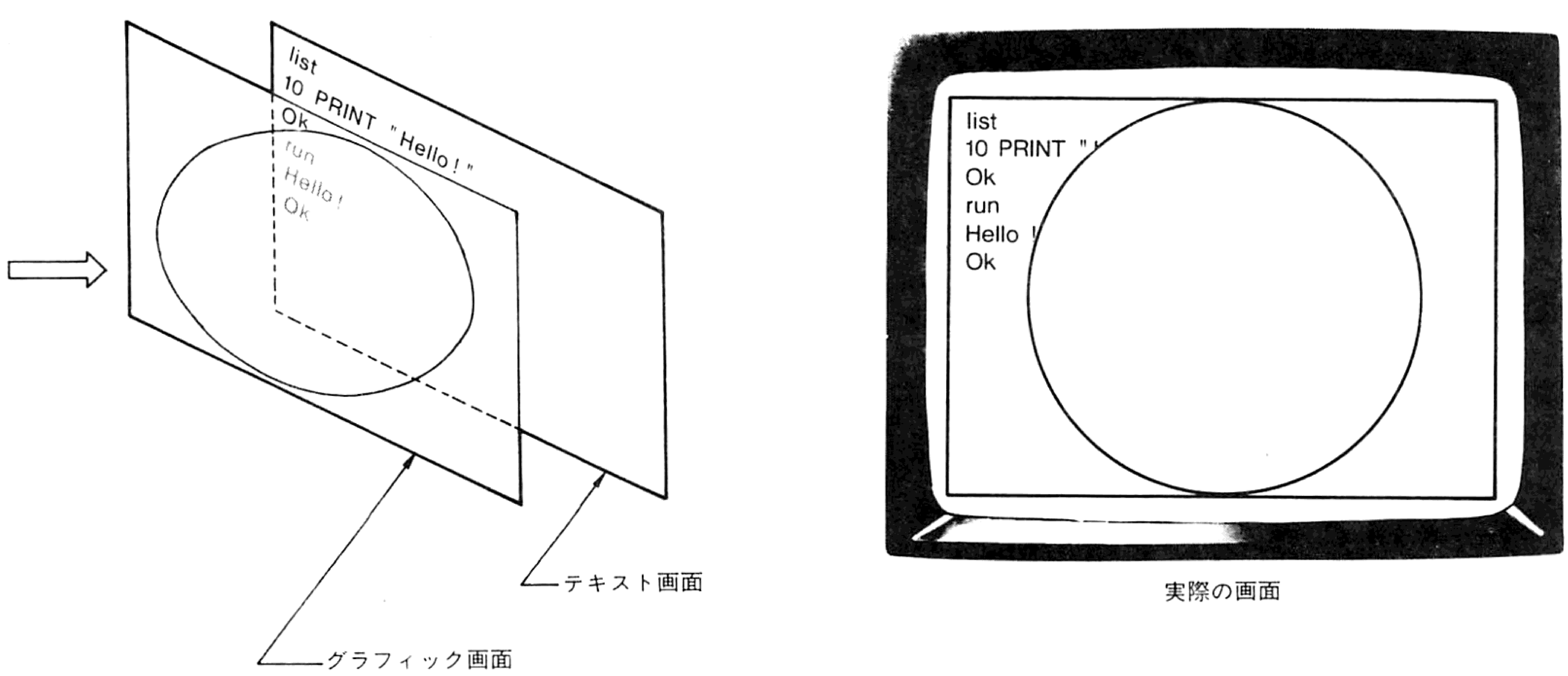
なお，カラーモードⅡではグラフィック画面2ページ間にも優先順位が付きます．（2章のSCREEN文参照）

優先順位の付け方を下図に示します．

カラーモードⅠ（テキスト画面とグラフィック画面1ページ）

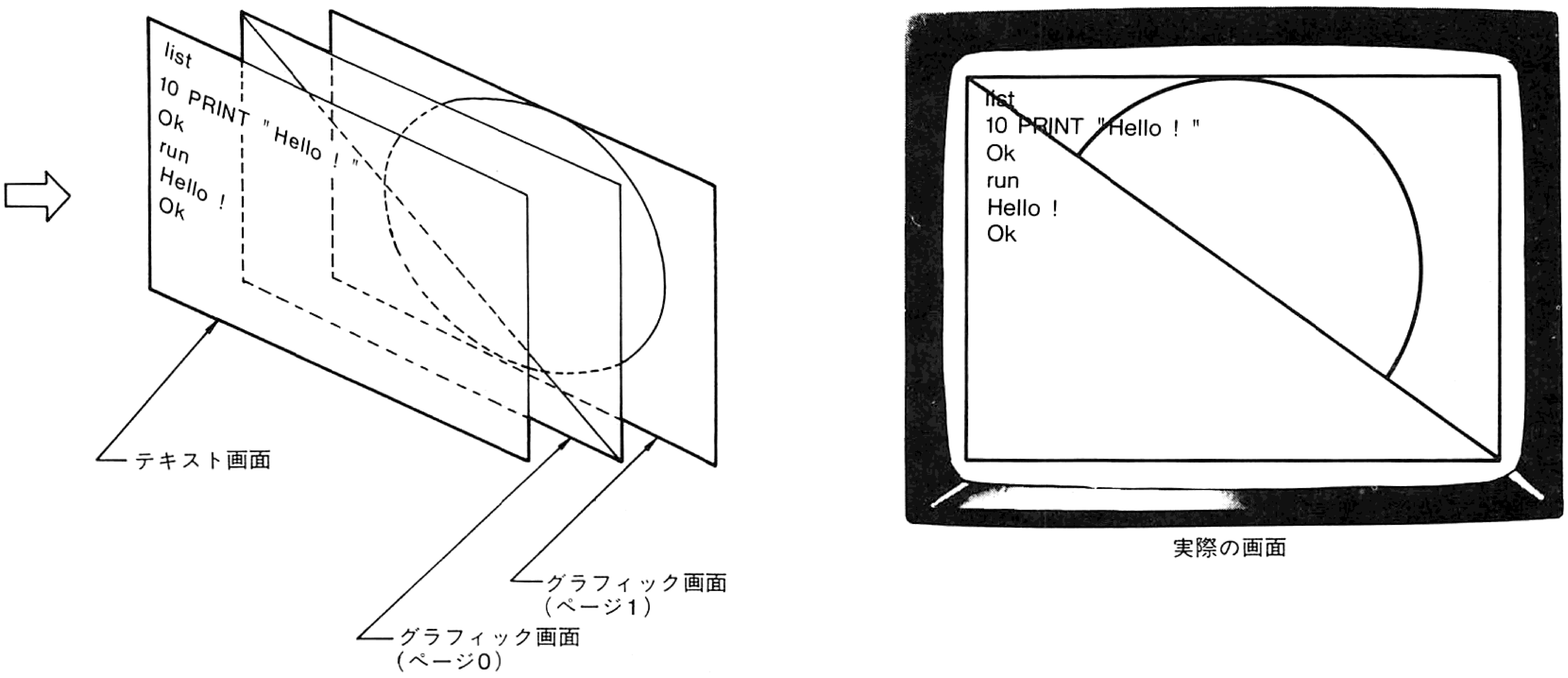


(i)　テキスト画面＞グラフィック画面

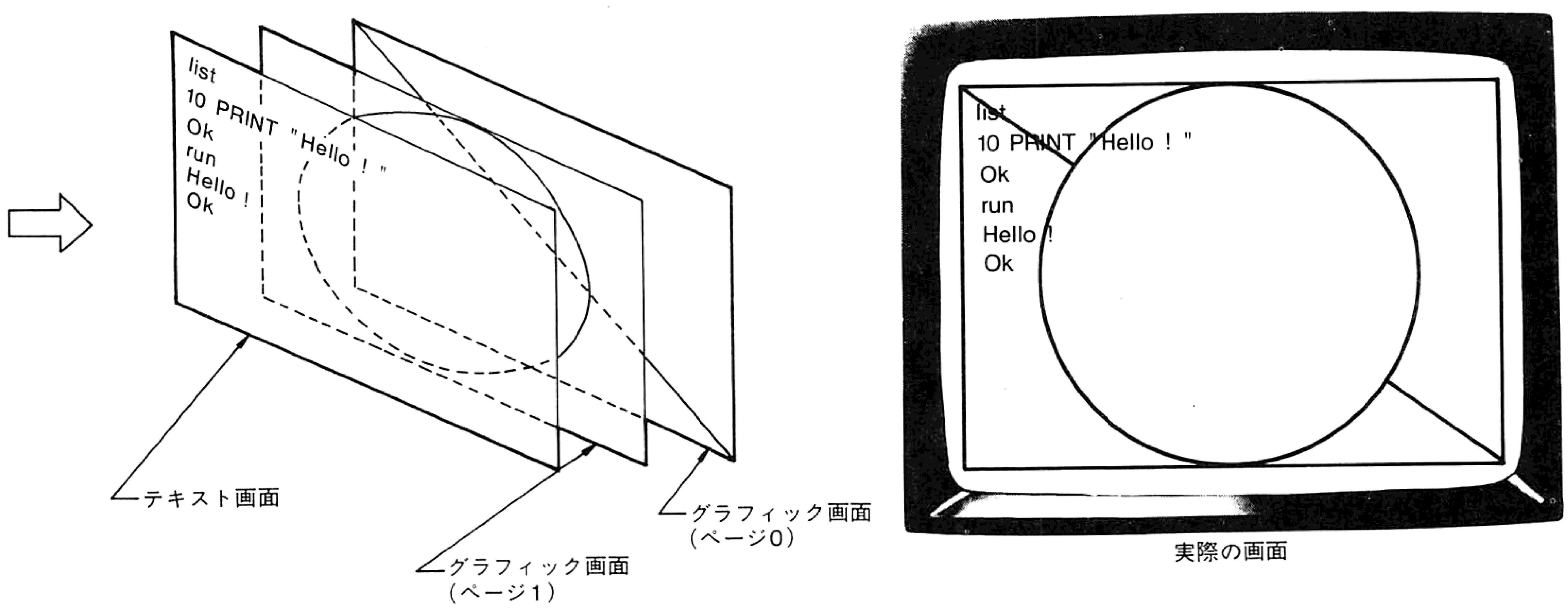


(ii)　グラフィック画面＞テキスト画面

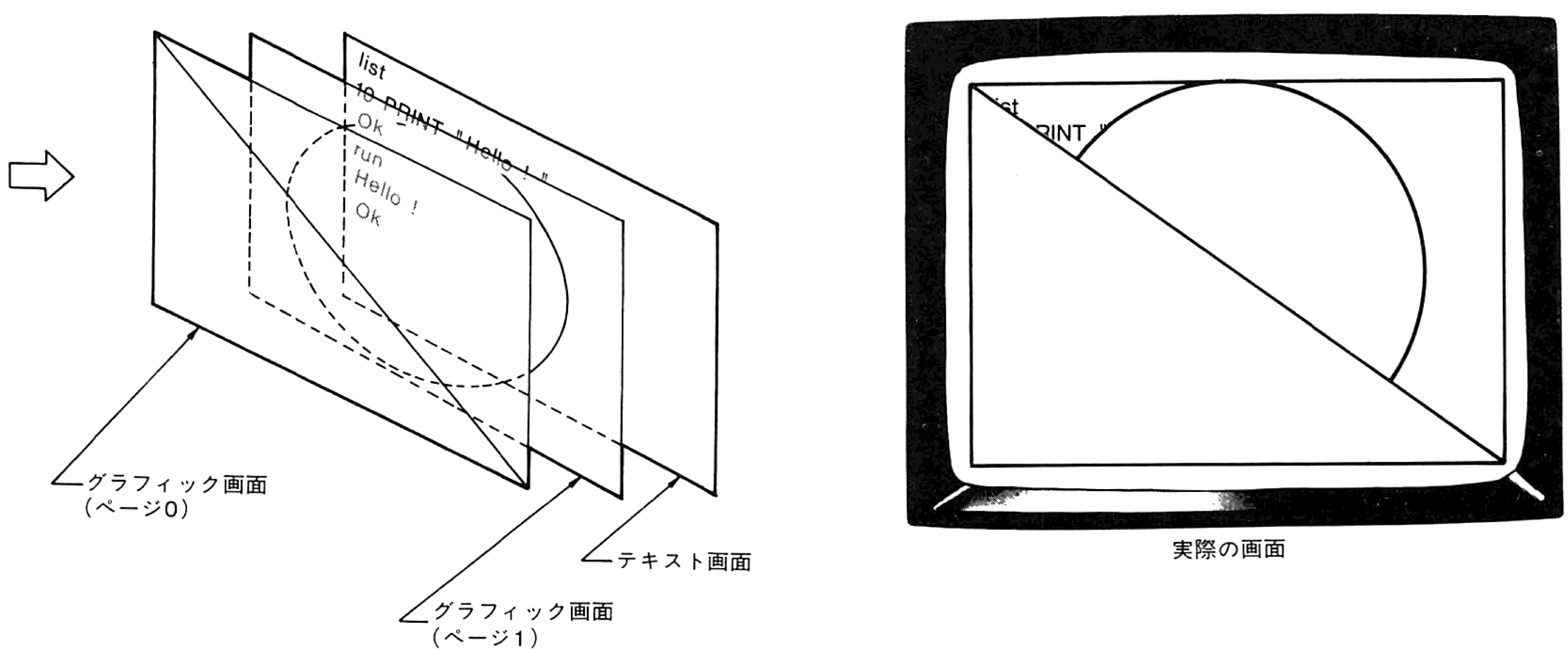
カラーモードⅡ(テキスト画面とグラフィック画面2ページ)



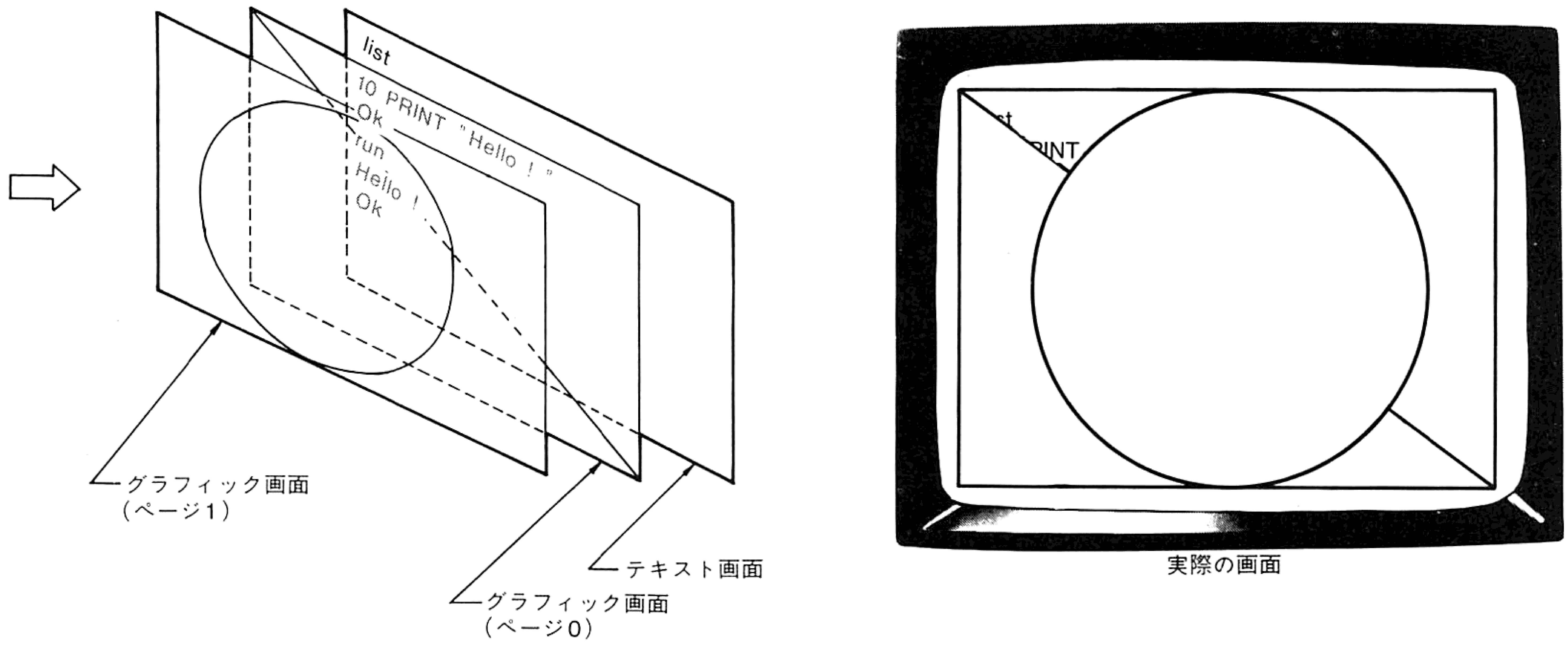
(i) テキスト画面>グラフィック画面(ページ0)>グラフィック画面(ページ1)



(ii) テキスト画面>グラフィック画面(ページ1)>グラフィック画面(ページ0)



(iii) グラフィック画面(ページ0)>グラフィック画面(ページ1)>テキスト画面



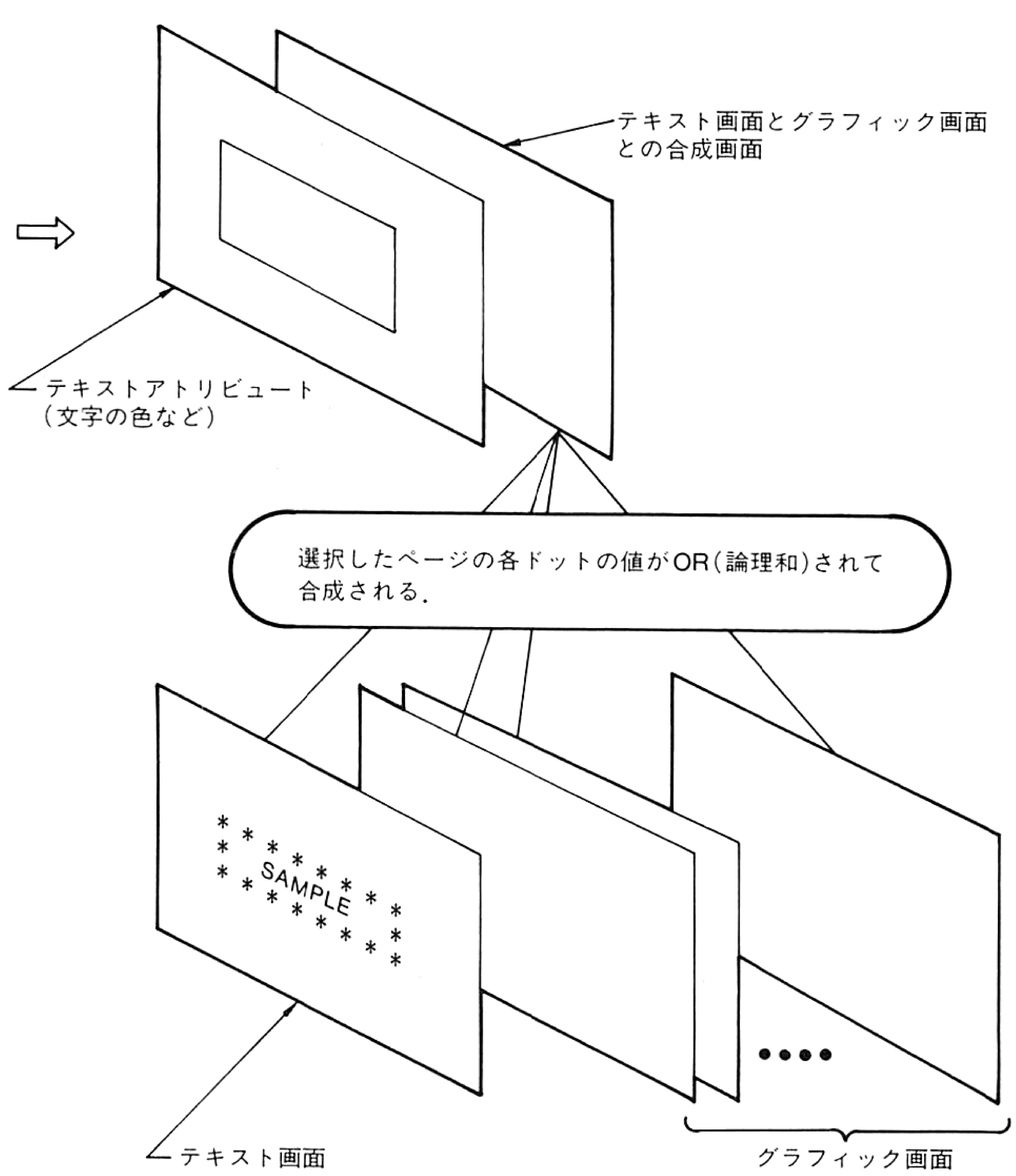
(iv)　グラフィック画面(ページ1)＞グラフィック画面(ページ0)＞テキスト画面

(2)　白黒モードでの画面合成

白黒モードでは,テキスト画面および選択したグラフィック画面のドットの論理和(OR)をとったものが表示されます．すなわち，少なくとも一つの画面にでも点があれば，それが表示されます．

さらに，テキスト画面のアトリビュート情報(文字の色，機能)がそれに重なり，キャラクタ単位に色やリバースなどの機能が設定できます．(下図参照)

白黒モード



1.14.4 アクティブページとディスプレイページ

PC-8001MKⅡSRは，カラーモードⅡ，白黒モードⅠ，Ⅱにおいて複数のグラフィック画面を持っています．

一般にグラフィック命令(LINE文，PAINT文，ROLL文など)は，一度にすべてのグラフィック画面に対してグラフィック操作を行うのではなく，1つのグラフィック画面に対してのみグラフィック操作を行います．この操作の対象となるグラフィック画面をアクティブページと呼んでいます．

アクティブページはSCREEN文の第3パラメータで指定します．指定するページの範囲は各画面モードによって異なります．

画面モード	ページ番号	ページ数
カラーモードⅡ	0～1	2
白黒モードⅠ	0～2	3
白黒モードⅡ	0～5	6

ディスプレイページとは，どのページを表示するかを指定するものです．カラーモードⅡ，白黒モードⅠ，Ⅱでは，単数あるいは複数のページを指定して表示することができます．

ディスプレイページの指定は，SCREEN文の第4パラメータで指定をします．詳しくは第2章のSCREEN文を参照してください．

1.15 ディスプレイ画面上の座標系

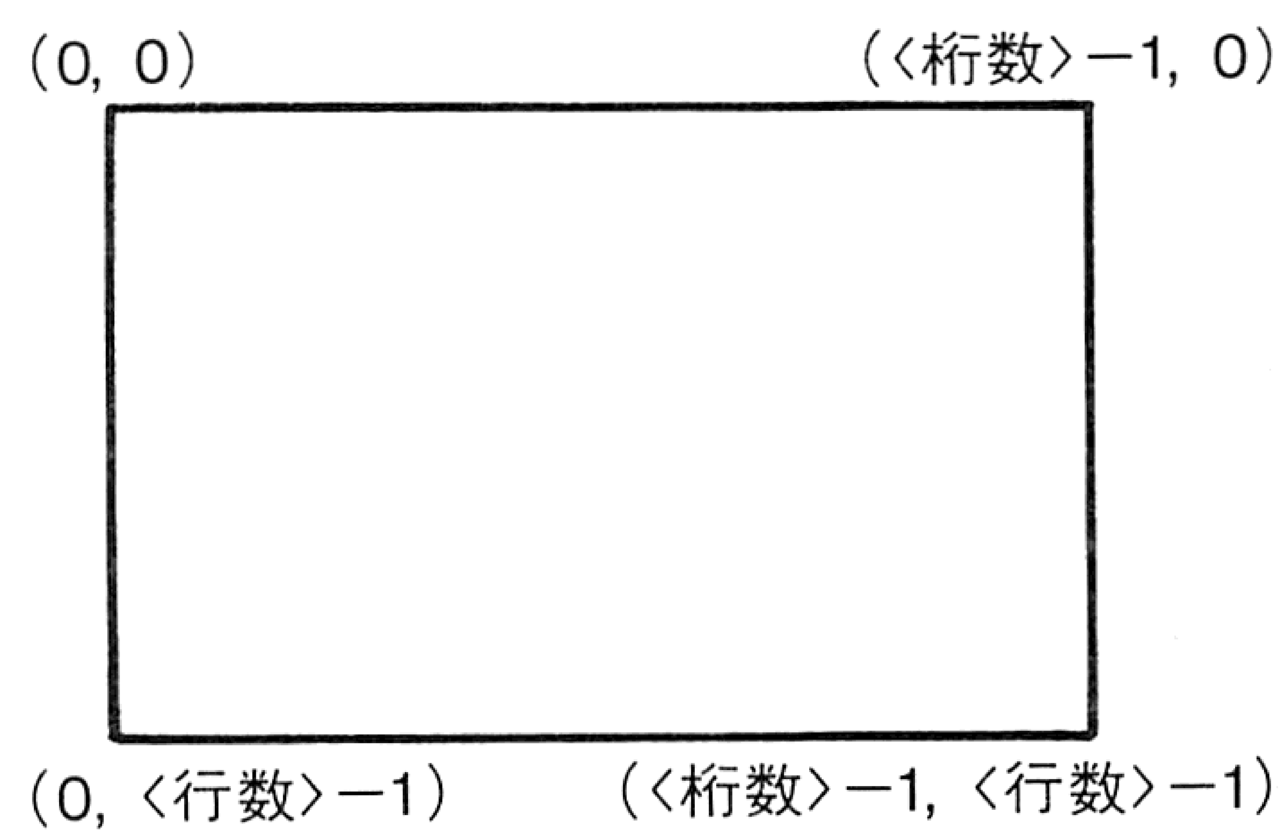
テキスト画面に文字を表示したり，グラフィック画面に点を描いたりするには画面上での位置を決める必要があります．この位置のことを座標といいます．

座標の決め方にはいろいろなものがありますが，一般にパーソナルコンピュータでは2次元の直交座標系を用います．

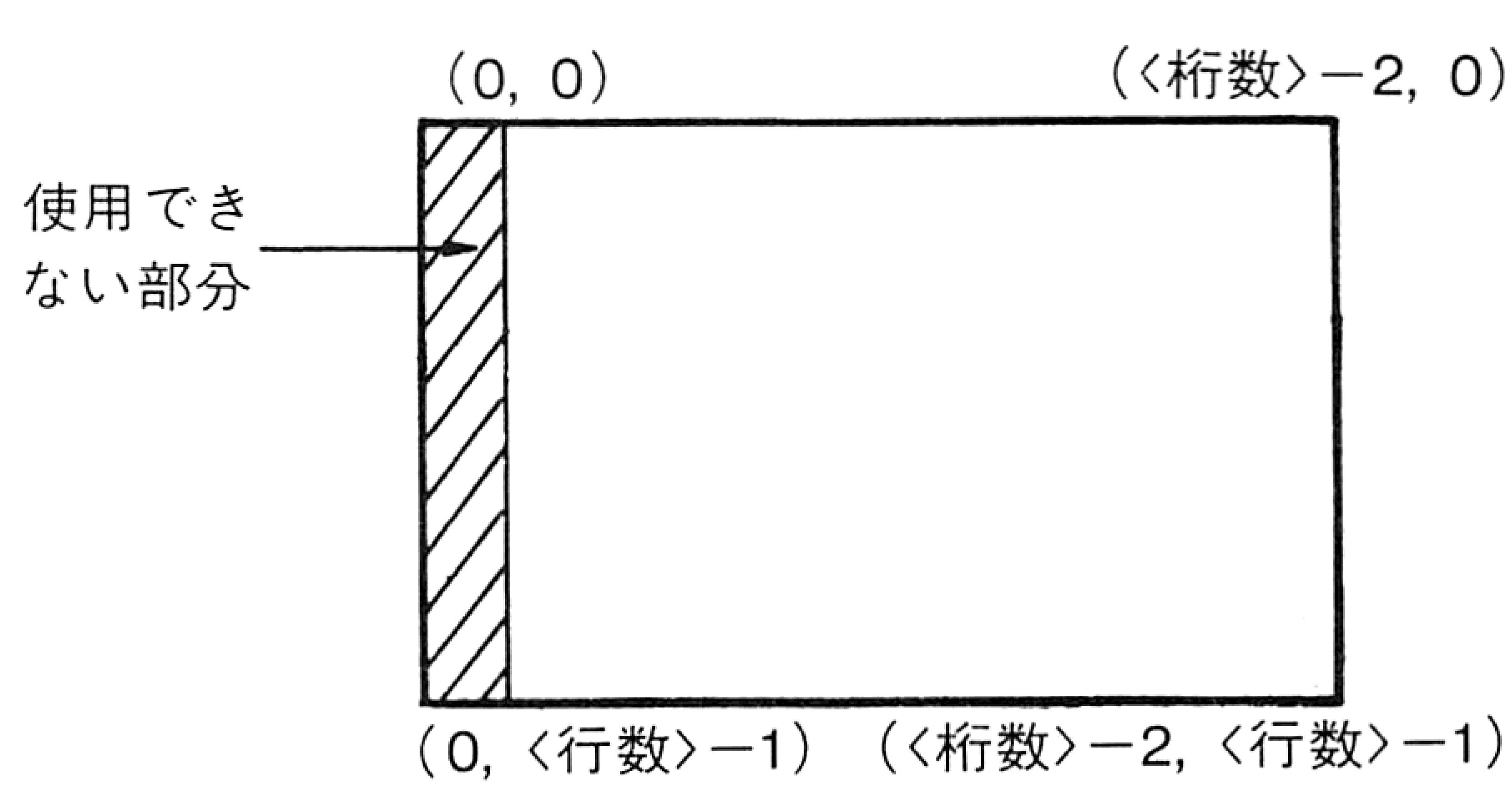
PC-8001MKⅡSRは，テキスト画面とグラフィック画面の2つの独立した画面を持っており，それぞれについて座標系が決められています．テキスト画面に対するものがキャラクタ座標系，グラフィック画面に対するものがグラフィック座標系です。ともに原点を画面左上とし，画面の横方向をX軸，縦方向をY軸とした2次元の直交座標系です．

1.15.1 キャラクタ座標系

テキスト画面に対して決められている座標系で，一文字の大きさが一つの単位となります．座標系の値の範囲は，WIDTH文によって設定される画面モードによって異なります．さらに，テキスト画面がカラーモードの場合は左端の縦一列が使用されませんので，桁数が1つ少なくなります．



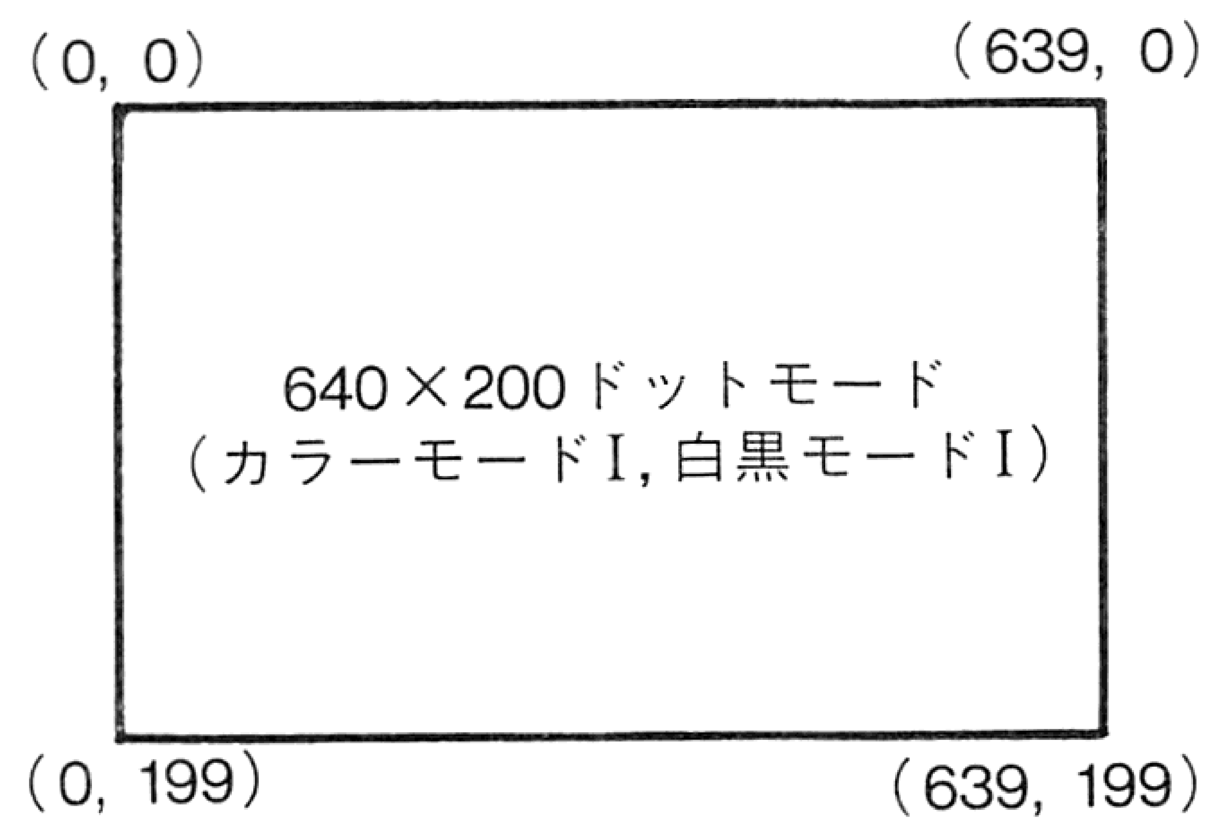
白黒モードの場合

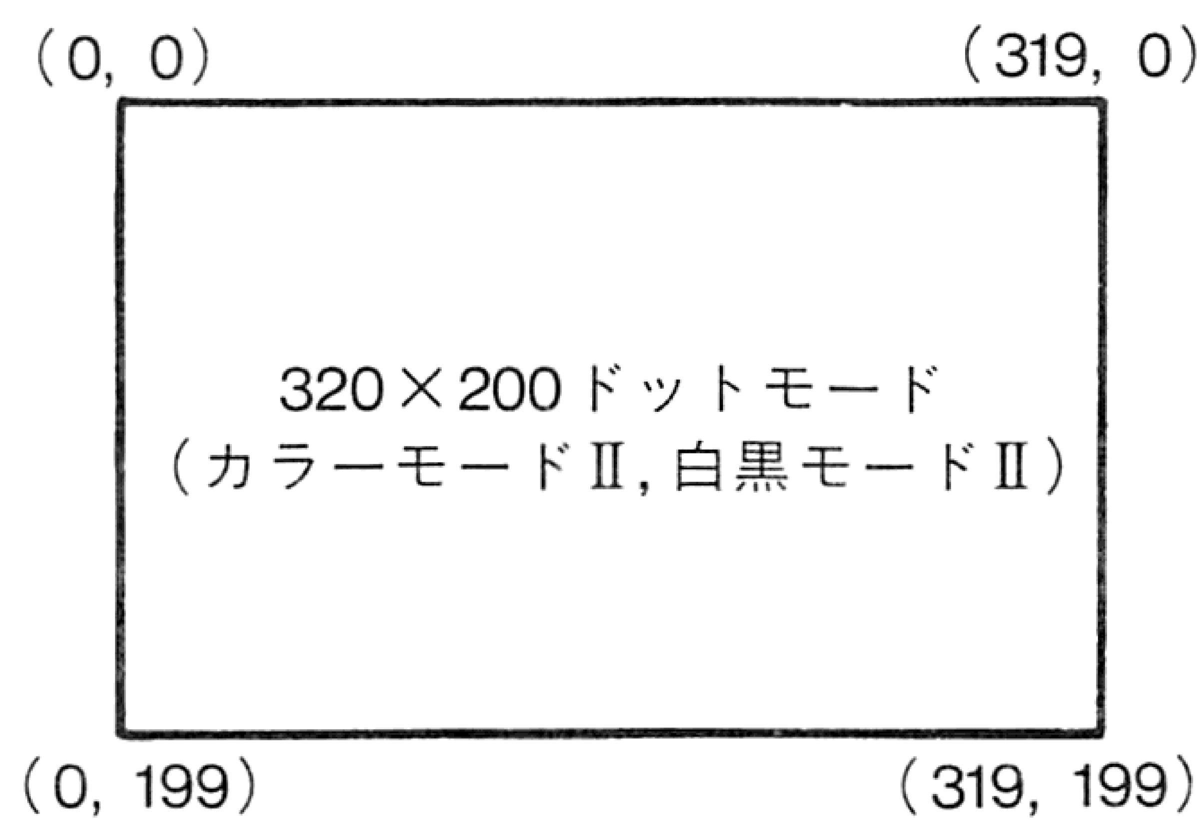


カラーモードの場合

1.15.2　グラフィック座標系

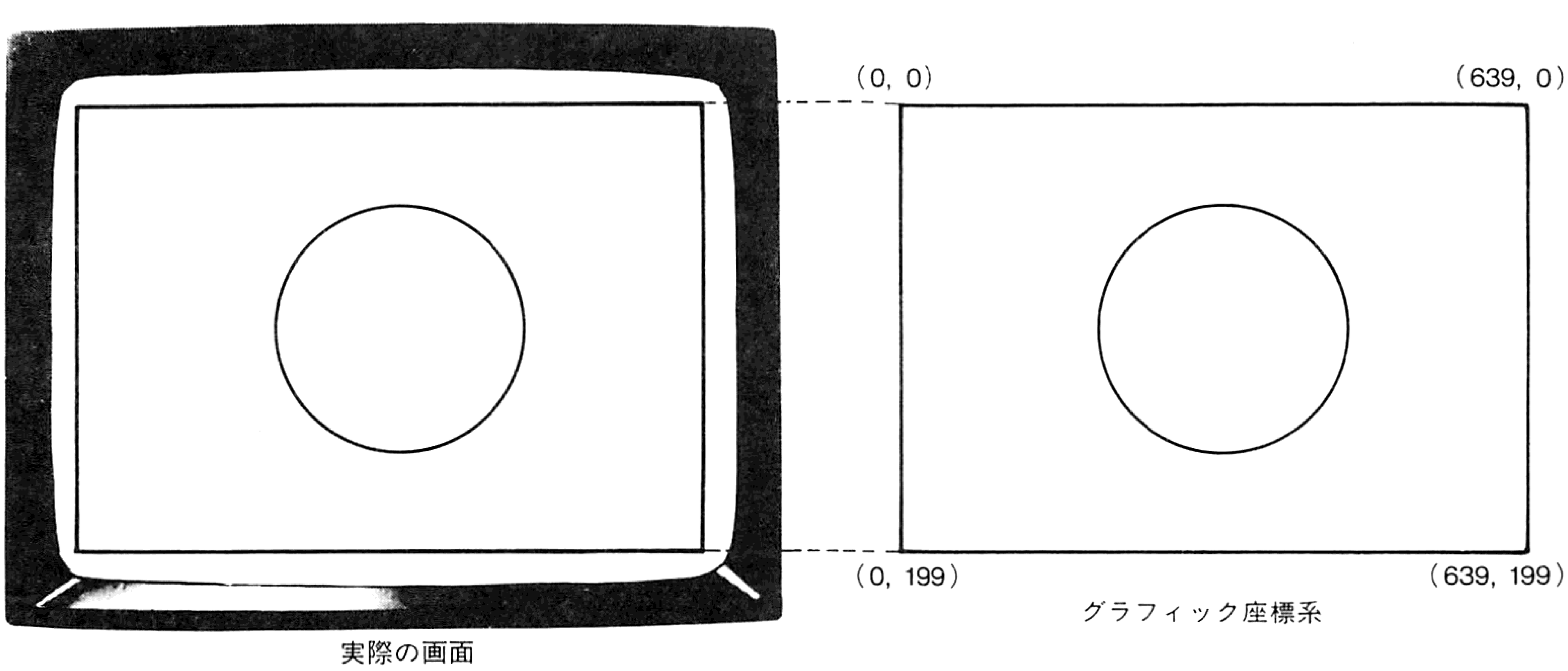
グラフィック画面に対して決められている座標系で，一つのドットが一つの単位となります．解像度の違いにより，次の2つの座標系があります．





1.16　ビューポート

システム起動時やSCREEN文(第1パラメータ指定)の実行後は，グラフィック座標と実際のグラフィック画面のドットは1対1に対応しています．



640×200ドットモードの場合

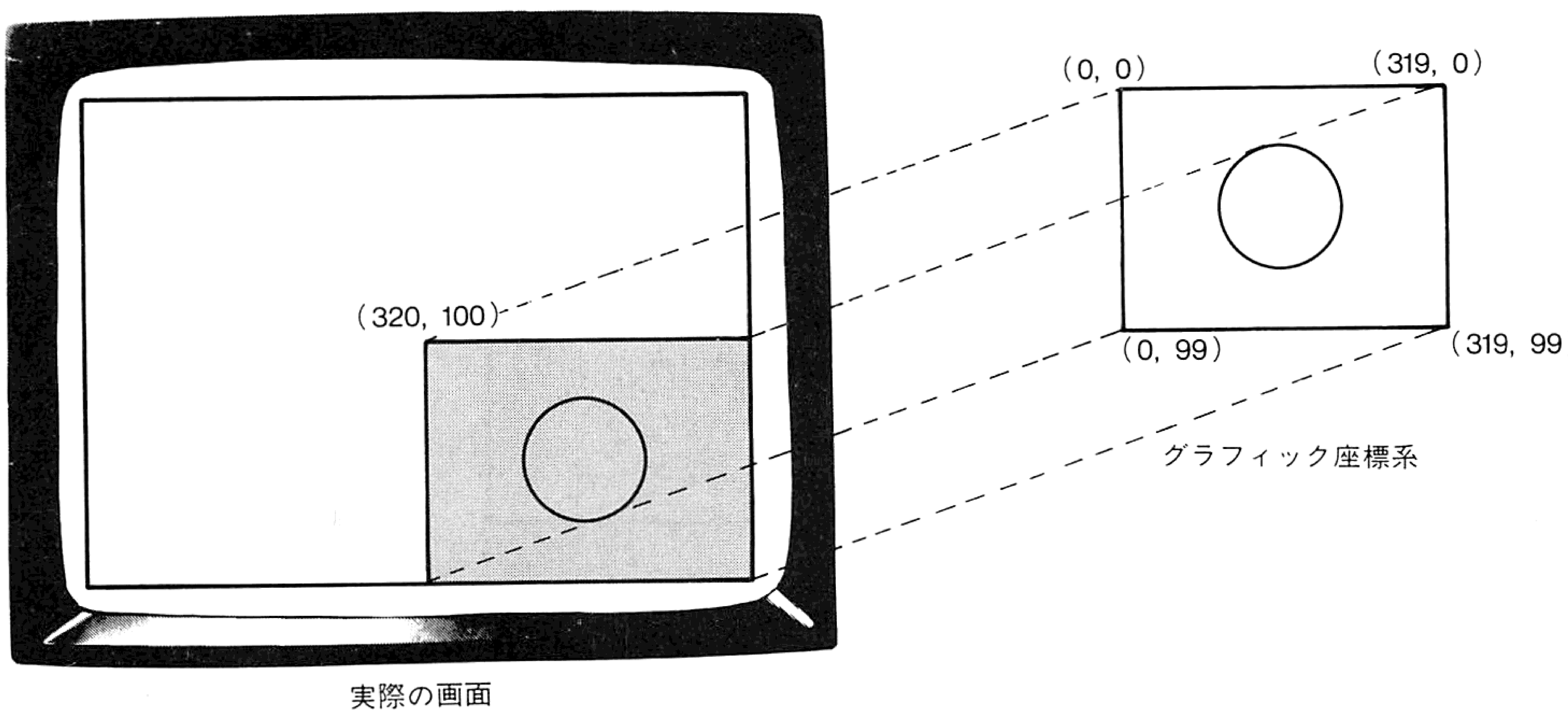
N_{80SR}-BASICでは，ビューポートという概念を取り入れて，このグラフィック座標とグラフィック画面の対応を変えることを可能にしています．

ビューポートとは，グラフィック命令の操作の対象となる(点や線を描く)グラフィック画面上での領域のことです．ビューポートはVIEW文によって設定します．

たとえば，(320，100)と(639，199)の2点を対角とする長方形の領域をビューポートとする場合は，次のVIEW文を実行します．

```
view(320,100)-(639,199)
```

カラーモードⅠで上のVIEW文を実行すると，グラフィック座標系と実際のグラフィック画面は次のように対応するようになります．



640×200ドットモードの場合

アミ(　　)の部分がビューポートで，このビューポートの左上の頂点がグラフィック座標の原点となります．つまり，グラフィック座標(0，0)が，実際のグラフィック画面上の(320，100)の点となるわけです．

すべてのグラフィック命令はビューポート内でのみ働きます．線などを描いたとき，ビューポートから外に出た部分については表示されません．

1.17　座標の指定の仕方

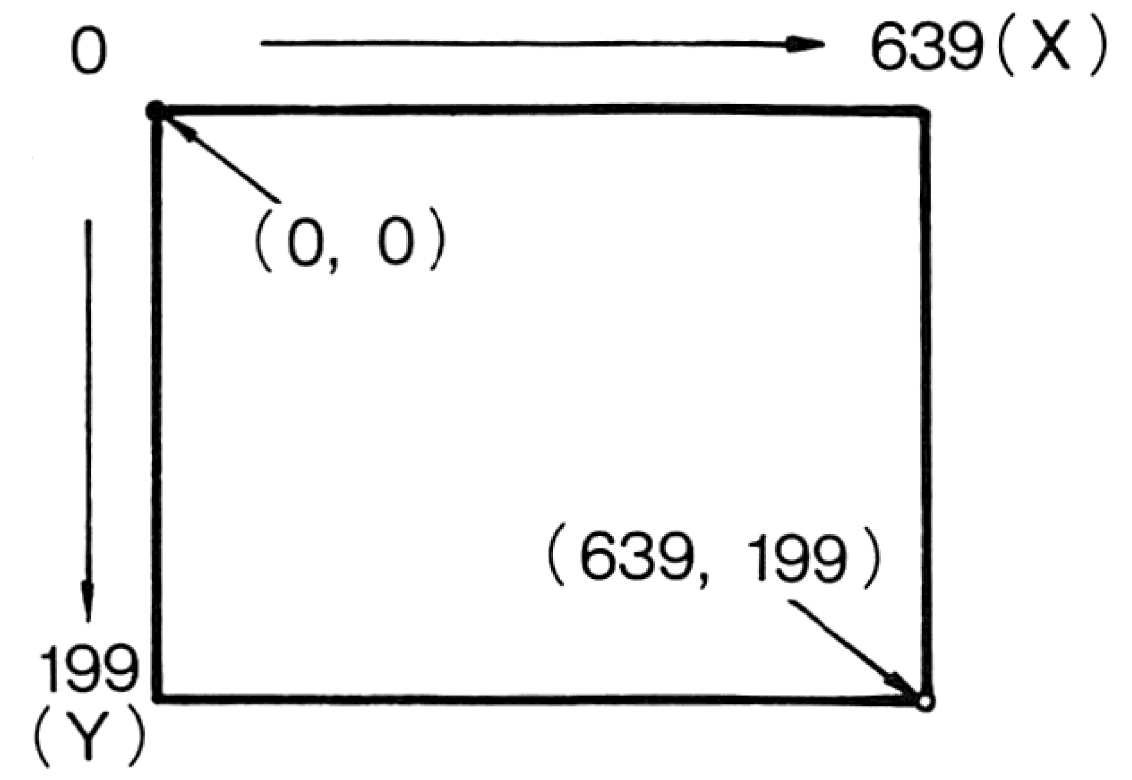
1.16で2つの座標系を説明しましたが，ここではこの2つの座標系に対して種々のグラフィック命令を使う場合の座標の指定の仕方を説明します．

最も一般的な座標の指定は"絶対座標"によるもので，キャラクタ座標，グラフィック座標の両方で用いられます．さらにグラフィック座標系では"相対座標"による指定も許されています．

次にこの2つの指定の仕方について説明します．

1.17.1　絶対座標による指定の仕方

絶対座標とは，X座標，Y座標が決まることによって一意的に決まる座標のことです．たとえば下図のような座標があったとすると，左上の頂点は常に(0，0)であり，右下の頂点は常に(639，199)です．“絶対座標による指定”とはこの絶対座標を使って位置を表す方法をいいます．



絶対座標(640×200ドットモードの場合)

これは最も一般的な指定法で，N_{80SR}-BASICが持つすべての座標系に対して有効です．2章で解説するグラフィック命令などにおいて座標を指定する場合，普通はこの方法により行います．たとえばPSET(gx，gy)という命令はグラフィック座標系内の1点にドットを描く命令ですが，これは次のように書くことができます．

例)　　pset(100,100)

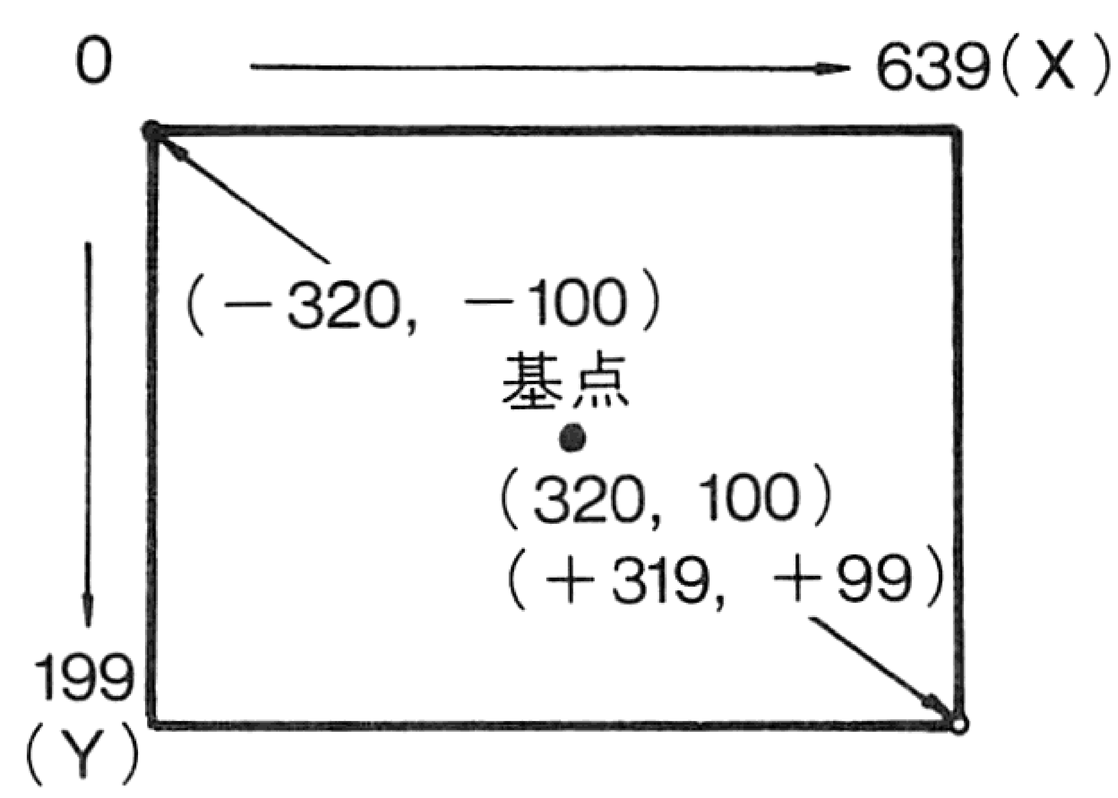
この場合，グラフィック座標の(100，100)にドットが描かれます．その他の命令の場合も，絶対座標による指定ではすべてこのように(gx，gy)という形になって表されます．

1.17.2　相対座標による指定の仕方

相対座標とは，座標系内のある1点を基点としたとき，X方向，Y方向にどれだけ相対的に移動するかによって決められる座標です．“相対座標による指定”とは，この相対座標を使って

位置を表す方法をいいます.

たとえば，次のような座標系があり，基点が(320，100)に設定されていたとすると，左上の頂点は(−320，−100)であり，右下の頂点は(＋319，＋99)となるわけです.



相対座標(640×200ドットモードの場合)

この指定法の優れている点は，相対移動量を表す(sx，sy)は常に同じ値であっても，基点の位置を動かすことによって違った位置を表現することができることです.

N_{80SR}-BASICはこの基点に当たるものとして，Last referenced Point(以後“LP”と略す)を持っています．このLPは最後に参照された座標という意味で，グラフィック命令で座標を与えた場合，最後に指定された座標を値として持ちます.

たとえばPOINT(gx，gy)という命令が実行された場合，LPは(gx，gy)という座標に設定されることになります(2章　POINT(ステートメント及び関数)参照).

グラフィック命令で相対座標を使って座標指定を行う場合は次のような書式になります.

例)　　pset step(sx，sy)

これはドットを描く命令ですが，その他の場合も同様で，座標を表す(sx，sy)の前に"STEP"を付けて(sx，sy)が相対量であることを表します.

グラフィック命令で相対座標の指定が許されているのは次に示す命令です(おのおのの命令の機能については2章で説明します).

PSET	PRESET
LINE	(1) PAINT
CIRCLE	(2) PAINT
GET@	POINT(ステートメント)
PUT@	

1.18 カラー

N_{80SR}-BASICでは，黒，青，赤，紫，緑，水色，黄色，白の8色を使うことができます．また，テキスト画面とグラフィック画面が分離しているため，個々にカラー指定を行えます．

(1) テキスト画面のカラー

テキスト画面のカラーは下記のカラーコードによって指定します．

カラーコード	0	1	2	3	4	5	6	7
色	黒	青	赤	紫	緑	水色	黄色	白

これらはキャラクタ単位で指定することができます．したがって，キャラクタ文字を1文字ずつ塗り分けることができますが，1行中で19回以上のカラー指定を行った場合，19回目以降は無視されます．

テキスト画面のカラーの設定については，2章のCONSOLE文，COLOR文を参照してください．

(2) グラフィック画面のカラー

640×200あるいは320×200の解像度を持つグラフィック画面では，ドット単位でカラーを指定することができます．これにより，ユーザは各ドットごとに8色の中から任意の1色を指定することができます．

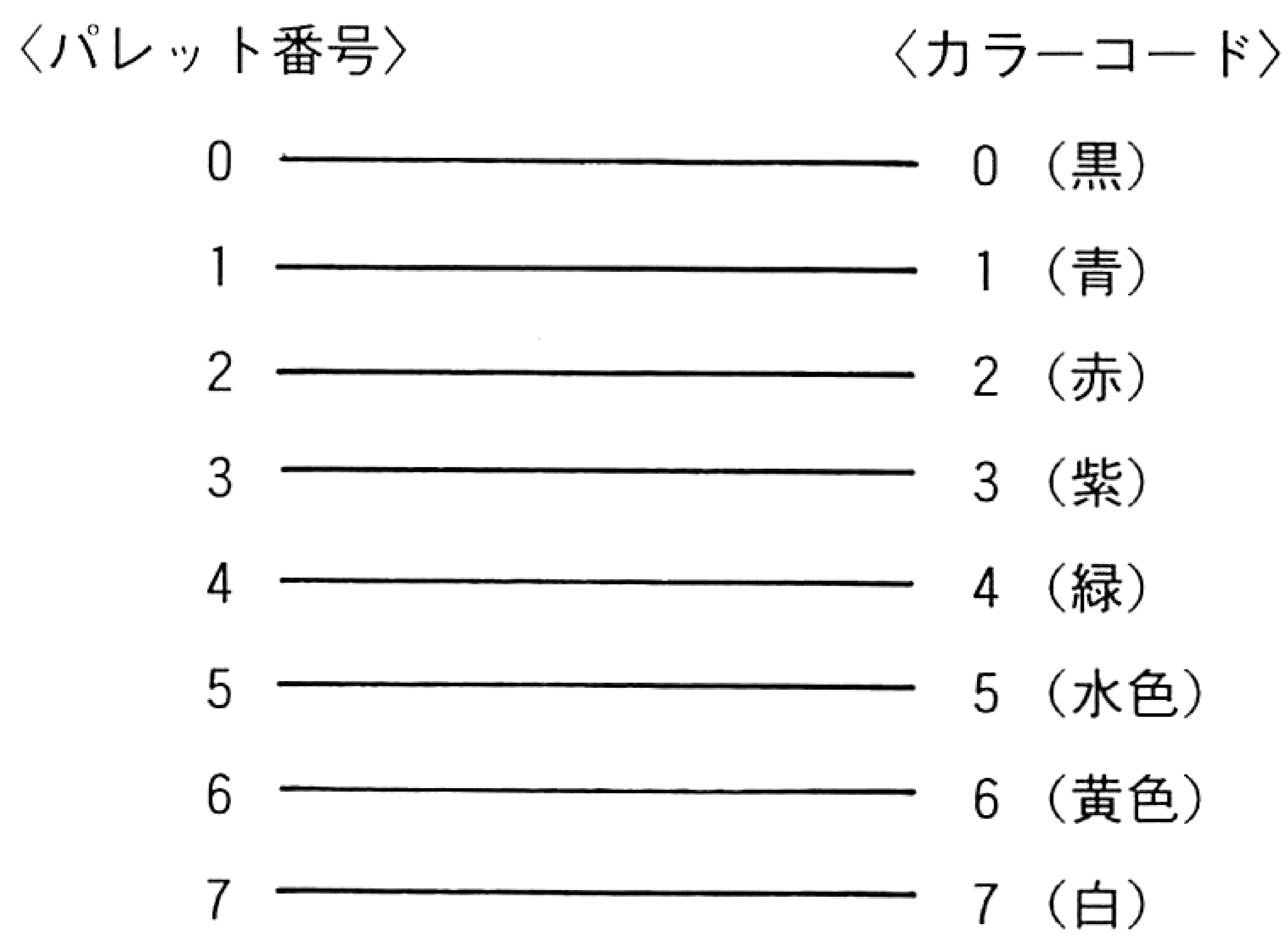
テキスト画面のカラー指定は"カラーコード"で行いましたが，グラフィック画面のカラー指定は"パレット"で行います．

(3) カラーコードとパレット番号

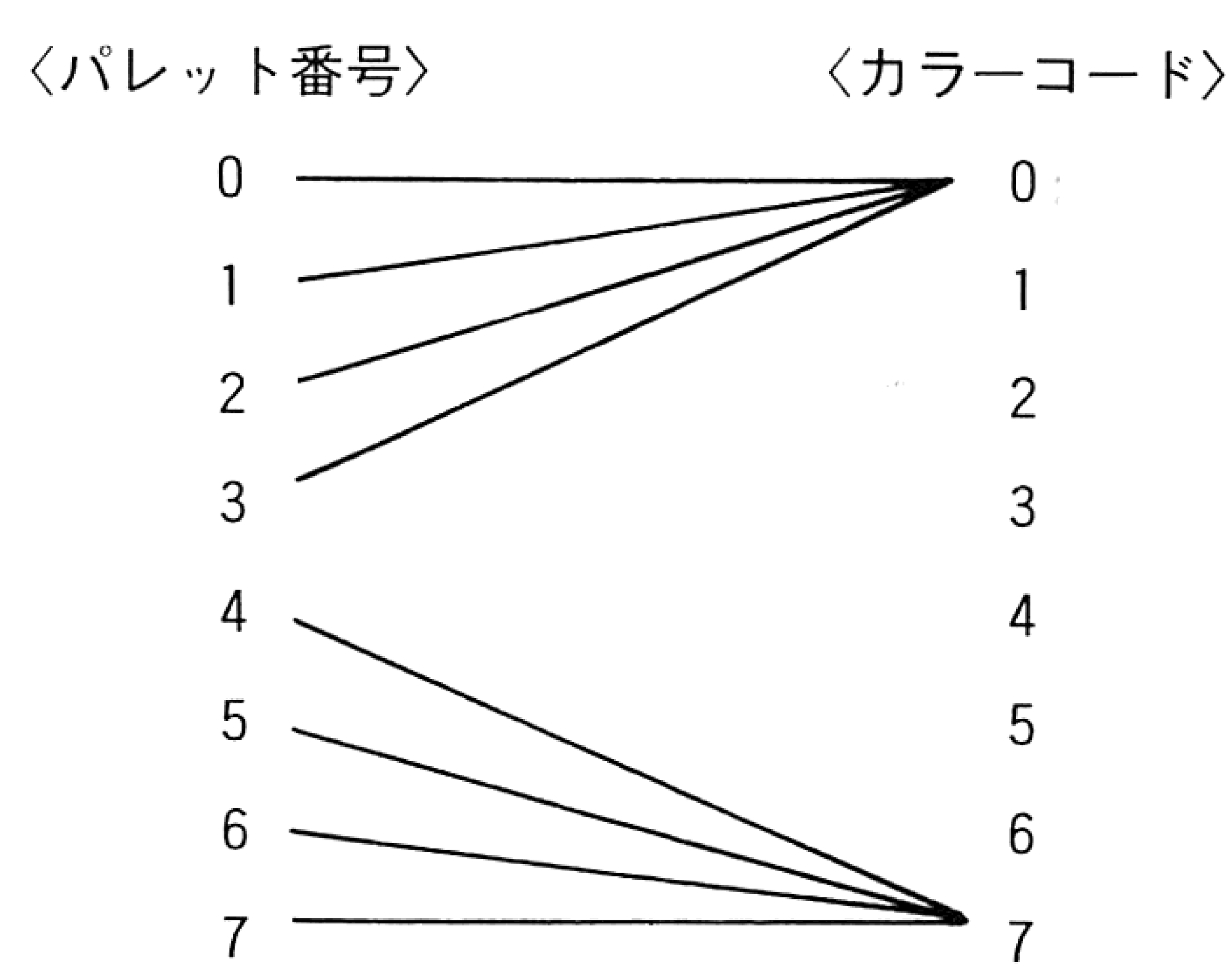
カラーコードとは色そのものを表すコード番号(0－黒，1－青，2－赤，…)ですから，融通性や汎用性に欠けています．たとえば色指定を1として青にしてしまったものは絶対的に"青"という意味を持ち，変更することができません．

そこで，グラフィック画面に対するカラー操作には"パレット"という概念が導入されています．"パレット"とは，その名の示すとおり絵を描くときに使うパレットという意味です．N_{80SR}-BASICでは0番から7番までの8つのパレットを持っていて，ユーザはこれらのパレットに好みの色を設定することができます．

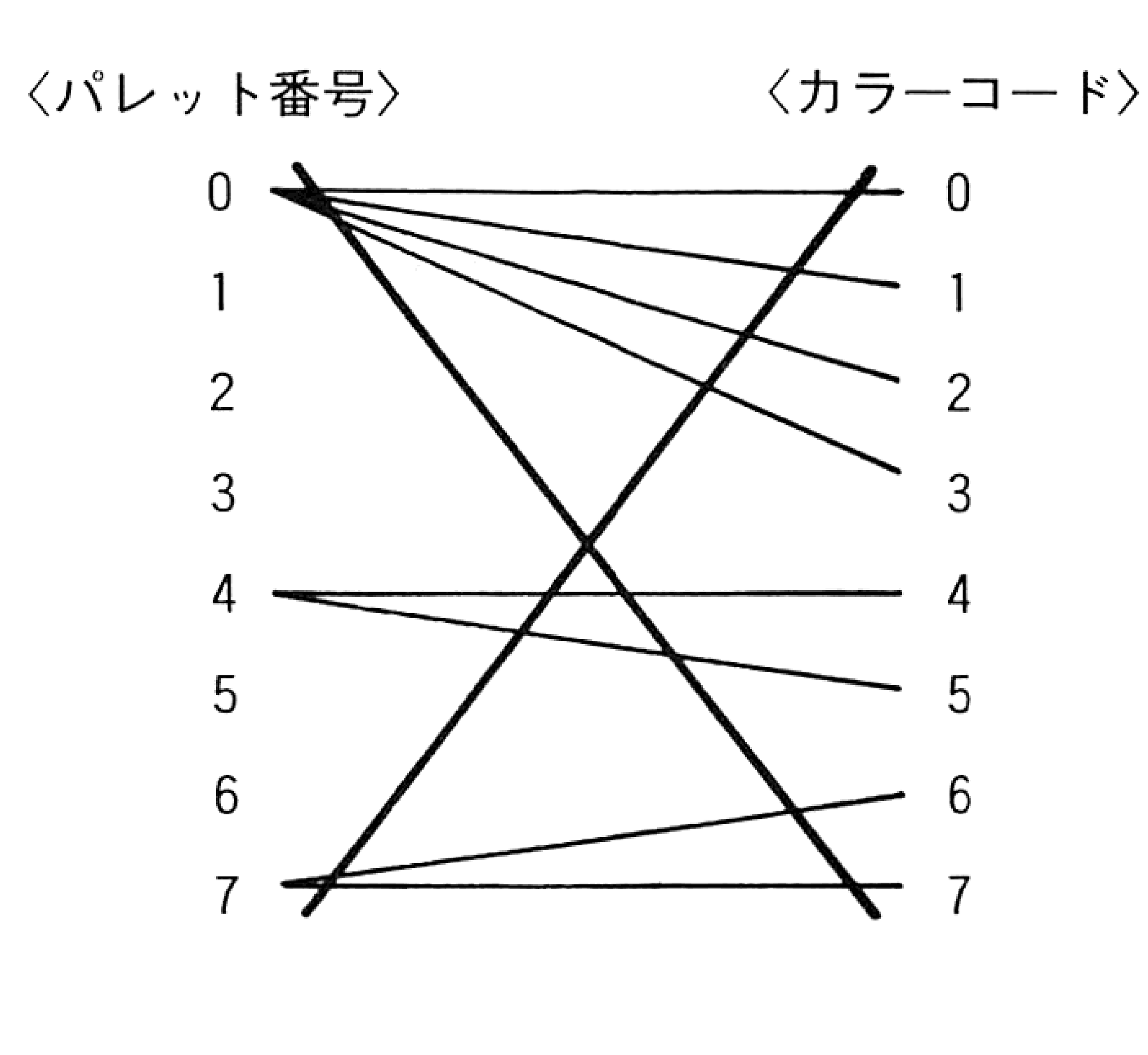
次の図はパレットの設定の一例で，パレット0が黒，パレット1が青，パレット2が赤，…というように対応させています．N_{80SR}-BASICが起動した最初の状態は，このように〈パレット番号〉=〈カラーコード〉になっています．



また次のように対応させると，パレットの0〜3は黒，4〜7は白となり，画面に表示される色は白と黒の2色ということになります．



ただし，次のように1つのパレットに複数の色を対応させて混合することは許されていません．



このようにしてパレットが設定されると，その瞬間から，グラフィック画面に表示されている色及びこれからグラフィック画面に描こうとする色はすべて変更されたパレットの影響を受けます．たとえば，最初パレット番号1に赤が設定されている状態で画面に線を描いたとします．次にそのパレット番号に青を設定したとすると，画面に描いてあった赤の線は瞬時に青の線になってしまうのです．

このようにパレットには決まった色という概念はなく，いつでも色を切り替えることができますから，プログラムに融通性があり，多彩な色の変化を実現することが可能です．

実際にパレットを設定するにはCOLOR＝文を用います．第1パラメータにパレット番号を指定し，第2パラメータにはカラーコードを指定します．詳しい説明は2章のCOLOR＝文を参照してください．

1.19　ファイルとファイル名

カセットテープやフロッピィディスク上にある意味のある情報(データ)の集まりをファイルと呼びます．

N_{80SR}-BASICで取り扱うことができるファイルには，プログラムファイル(BASICプログラムや機械語プログラムが入っている)やデータファイル(データが入っているファイル)があります．なお，BASICプログラムをアスキーセーブしたものはデータとして入出力ができるので，データファイルにもなります．

個々のファイルは名前で管理され，この名前をファイル名と呼びます．ファイル名は次の形式から成る文字列で定義されます．ただし，カセットテープのデータファイルにはファイル名はありません．

• カセットテープの場合

〈ファイル名〉
└------ 1文字以上6文字以内の文字列

例）　csave "demo"
　　　cload "graph"

• フロッピィディスクの場合

〔〈ドライブ番号〉:〕〈ファイル名〉〔.〕〔〈拡張子〉〕

- 〈拡張子〉---- 3文字以内の文字列(省略可)
- 〈ファイル名〉---- 1文字以上6文字以内の文字列
- 〈ドライブ番号〉---- フロッピィディスクの入っているドライブ番号で，省略した場合は1とみなされます.

〈ファイル名〉および〈拡張子〉の中にキャラクタコード0〜31の文字を含むことはできません. また，〈ファイル名〉の最初の1文字はキャラクタコード255の文字以外を使います.

例)

```
save "demo.nsr"
open "2:telphn" as #1
```

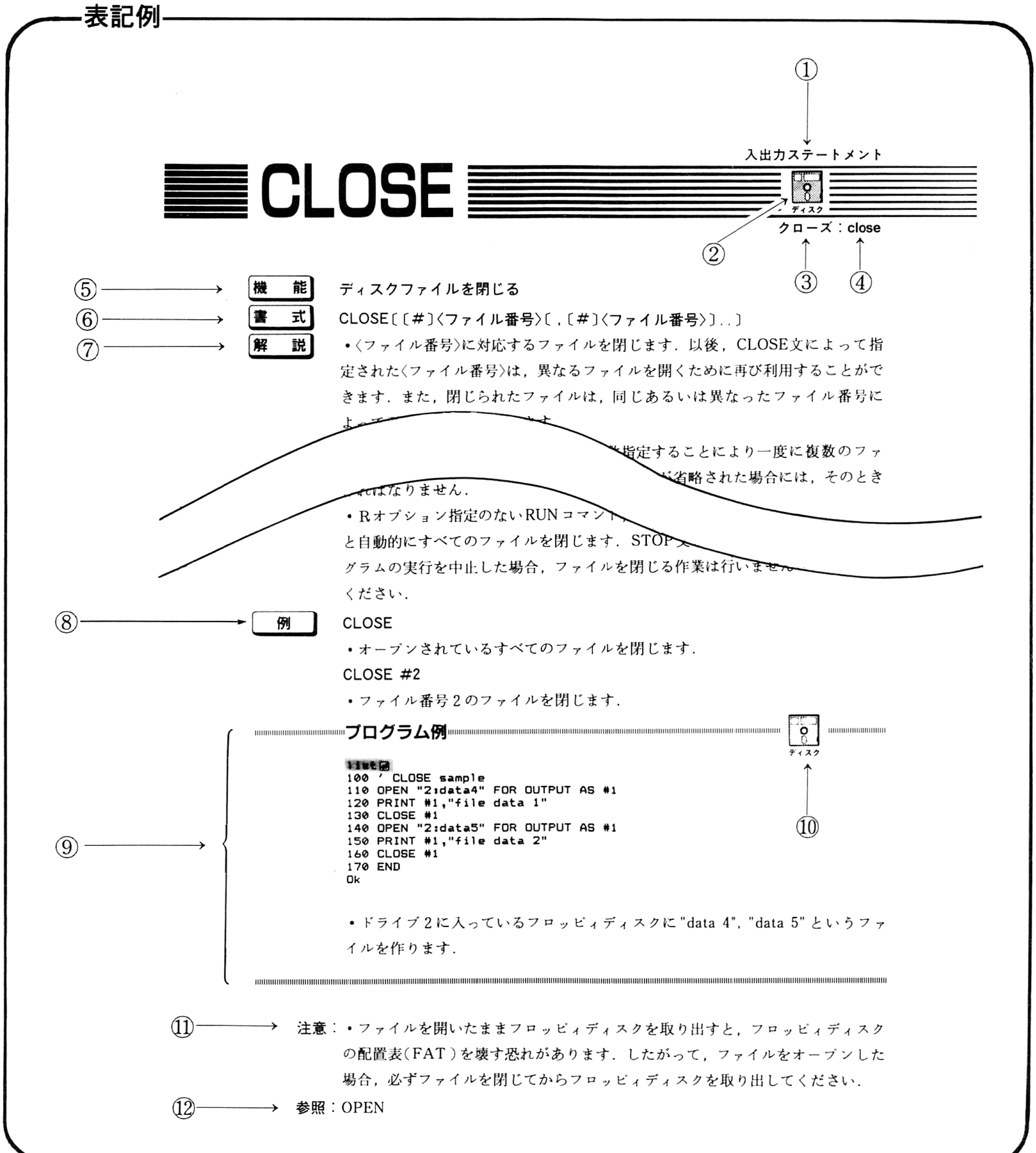
第2章

コマンド・ステートメント・関数・予約変数

2章の見方

N$_{80SR}$-BASICのすべてのコマンド，ステートメント，関数，予約変数を2章で解説します．CLOSEというステートメントを解説したページを例にあげて，使われている記号や表記方法を説明します．

表記例



①

入出力ステートメント

CLOSE

②ディスク

クローズ：close

③ ④

⑤ 機能　ディスクファイルを閉じる

⑥ 書式　CLOSE〔〔#〕〈ファイル番号〉〔,〔#〕〈ファイル番号〉〕..〕

⑦ 解説　•〈ファイル番号〉に対応するファイルを閉じます．以後，CLOSE文によって指定された〈ファイル番号〉は，異なるファイルを開くために再び利用することができます．また，閉じられたファイルは，同じあるいは異なったファイル番号によっ

…指定することにより一度に複数のファ

…省略された場合には，そのとき

…ればなりません．

•Rオプション指定のないRUNコマン…

と自動的にすべてのファイルを閉じます．STOP…

グラムの実行を中止した場合，ファイルを閉じる作業は行いませ…

ください．

⑧ 例　CLOSE

•オープンされているすべてのファイルを閉じます．

CLOSE #2

•ファイル番号2のファイルを閉じます．

⑨ **プログラム例**　⑩ディスク

```
list
100 ' CLOSE sample
110 OPEN "2:data4" FOR OUTPUT AS #1
120 PRINT #1,"file data 1"
130 CLOSE #1
140 OPEN "2:data5" FOR OUTPUT AS #1
150 PRINT #1,"file data 2"
160 CLOSE #1
170 END
Ok
```

•ドライブ2に入っているフロッピィディスクに"data 4"，"data 5"というファイルを作ります．

⑪ 注意：•ファイルを開いたままフロッピィディスクを取り出すと，フロッピィディスクの配置表（FAT）を壊す恐れがあります．したがって，ファイルをオープンした場合，必ずファイルを閉じてからフロッピィディスクを取り出してください．

⑫ 参照：OPEN

①機能別分類

BASIC上における機能を示します．その機能を大きく分けると次の4つに分類されます．

(1) コマンド

プログラムの実行や編集などを指示する命令です．一般的にダイレクトモードで使用します．

(2) ステートメント

単独である動作を行う命令です．主にプログラム中で使用します．(コマンドとステートメントは必ずしも明確に分かれていません．)

(3) 関数

ある値(引数)を引き渡して処理された結果を，値として返す機能を持ちます．単独で使われることはなく，代入文やコマンド，ステートメント中で使われます．

(4) 予約変数

BASICが専用に使っている変数です．

さらにこのマニュアル中では，これらの機能を次のように分類しています．

- **一般コマンド，一般ステートメント**

  プログラムを作る，動作させる，停止させる，変数などの定義をするといった基本的な動作や，プログラムの流れのコントロール(繰り返しや分岐)を行います．

- **入出力コマンド，入出力ステートメント，入出力関数**

  フロッピィディスクユニット，カセットテープレコーダ，画面，キーボード，プリンタ，スピーカ，RS-232C通信ポートなどの操作を行います．

- **画面制御ステートメント，画面制御関数，画面制御予約変数**

  テキスト画面およびグラフィック画面の初期化やテキスト画面に対する操作を行います．

- **グラフィックステートメント，グラフィック関数**

  グラフィック画面に対する操作を行います．

- **ミュージックステートメント**

  音楽演奏やシンセサイザーICの操作を行います．

- **数値関数**

  数値演算を行います．

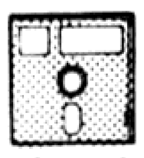
- **文字ステートメント，文字関数，文字予約変数**

  文字列の操作を行います．

- **特殊コマンド・特殊ステートメント・特殊関数・特殊予約変数**

  機械語モニタに入るとき，エラー処理，機械語や入出力ポートの操作などの特殊な働

きをします.

②  ディスク このマークが付いている命令はDISKバージョンでサポートされています. したがって, ROMバージョンで使うことはできません.("Disk BASIC Feature"エラーとなります.)

③ 命令の読み方を表します.

④ 命令をフルネームで示します.

⑤ **機　能** 命令の機能を簡単に説明します.

⑥ **書　式** 命令の記述の仕方を示します. 実際に入力を行う場合は次のような決まりに従ってください.

1　カギカッコ〈,〉に囲まれていない文字や記号はそのまま入力します. 入力する文字は大文字でも小文字でもかまいません.

ただし, 次の場合は大文字と小文字を区別しなければなりません.

(1)　ダブルクォーテーション(")で囲まれた文字列は大文字と小文字を区別します.

例)　"dskut5.nsr" ←――ファイル名

set 1, "P" ←――属性文字

(2)　数値を指数形式で表す場合に用いるEとDは大文字でなければなりません.

例)　A!=−7.09E−06

B#=1.094328564D5

2　カギカッコ〈,〉で囲まれた項目は, ユーザが具体的に指定します. 項目の主な種類は次のとおりです.

〈数式〉:　数値定数や数値変数を単独で, あるいは演算子で結合したものです.

例)　1, 10, 1/SIN(A!), B2, C＊D

〈文字列〉:　文字定数あるいは文字型変数を表します.

例)　"ABC", ST$

〈式〉:　〈数式〉または〈文字列〉を表します.

〈機能〉, 〈スイッチ〉:

0, 1, 2, 3などの整数で, コマンドやステートメントの機能を指定するものです.

〈ファイル名〉:　カセットテープやフロッピィディスク上のファイルを表す名前です. カセットテープの場合"ファイル名"の形式を, フロッピィディスクの場合"ドライブ番号:ファイル名. 拡張子"の形式を持ちます.

例)　　"2:sysgen.nsr","demo1"

〈ファイル番号〉：データファイルをオープンするときに定義します．フロッピィディスクを使用する場合は，N_{80SR}-BASIC DISKバージョンを起動したときに設定した数の範囲内で指定します．カセットテープを使用する場合は，データファイルをオープンする必要はなく，ファイル番号には−1を指定します．

〈ドライブ番号〉：フロッピィディスクユニットの各々のドライブに割り当てられる番号です．

5.25インチフロッピィディスクユニット				8インチフロッピィディスクユニット			
PC-80S31 PC-8031-2W PC-8031-1W		PC-80S32 PC-8032-2W PC-8032-1W		PC-8881 PC-8881(1)		PC-8882 PC-8882(1)	
drive 1	drive 2	drive 3	drive 4	drive 1	drive 2	drive 3	drive 4
1	2						
1	2	3	4				
				1	2		
				1	2	3	4
3	4			1	2		
3	4	5	6	1	2		
5	6			1	2	3	4
5	6	7	8	1	2	3	4

表中の1〜8までの番号が割り当てられるドライブ番号です．空白は，フロッピィディスクユニットが接続されていないことを表します．

〈カラーコード〉：画面に表示可能な8色の色につけられた番号です．

0：黒　　1：青　　2：赤　　3：紫
4：緑　　5：水色　6：黄色　7：白

3　角カッコ〔，〕で囲まれた項目は，省略することができます．省略した場合，デフォルト値(BASICによって設定される値)または以前に指定した値が適用されます．

4　省略記号(...)の続く項目は，1行の許す長さ(255文字)内で繰り返すことができます．

例)　〈定数〉〔,〈定数〉...〕の場合　0,10,15

5　座標指定については次のように表しています．

(gx,gy)…グラフィック座標

(cx,cy)…キャラクタ座標

(sx , sy)… 相対座標

⑦ **解　説** 命令の使用方法や詳しい機能，それに関して注意しなければならない点などを説明します.

⑧ **例** 実際の入力の見本として簡単な例とそれについての解説を示します.
命令が関数の場合は，**返される値**⇒という記号を使って処理された結果も同時に示します.

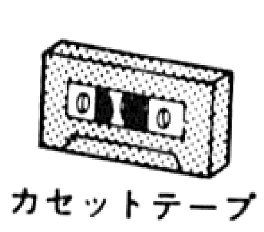
⑨ **プログラム例**または**実　行　例**

プログラム例には，その命令といくつかの文を混えたプログラムとその実行結果，および解説が示されています．実行例には，その命令の実際の使い方と解説が示されています.

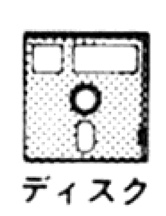
アミ（　　）の部分はユーザが実際に入力する部分で，記述されているとおりに入力した場合は，続く実行結果と同じになります．ただし，システムの構成が違う場合など必ずしも同じ実行結果にならないことがあります.

⑩ システム構成

プログラム例あるいは実行例を実行する際，本体とディスプレイの他に必要な機器を示します．以下のマークが付いていない場合は，本体とディスプレイのみで実行できます.



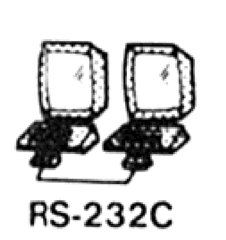
… カセットテープレコーダ／カセットテープ



… フロッピィディスクユニット／フロッピィディスク



… プリンタ／用紙



… RS-232Cに接続される機器（他のパーソナルコンピュータなど）／RS-232Cケーブル

⑪ **注意**：特に注意してほしい点を示します.

⑫ **参照**：関連の深い項目を示します.

ABS

アブソリュート：absolute

機能 絶対値を返す

書式 ABS（〈数式〉）

解説
- 〈数式〉の値の絶対値を返す関数です．
- ABS関数の演算結果は，〈数式〉に倍精度実数が含まれる場合は倍精度に，その他の場合は単精度になります．

例

ABS(−2.4) 返される値 ⇒ 2.4

ABS(2.4) 返される値 ⇒ 2.4

ABS(0) 返される値 ⇒ 0

プログラム例

```
list↵
100 ' ABS sample
110 INPUT "テン 1 ノ ザヒョウ ハ: x,y ";X1,Y1
120 INPUT "テン 2 ノ ザヒョウ ハ: x,y ";X2,Y2
130 D=SQR(ABS(X1-X2)^2+ABS(Y1-Y2)^2)
140 PRINT:PRINT "テン 1 ト テン 2 ノ キョリ ハ ";D
150 END
Ok
run↵
テン 1 ノ ザヒョウ ハ: x,y ? 0,0↵
テン 2 ノ ザヒョウ ハ: x,y ? 100,0↵

テン 1 ト テン 2 ノ キョリ ハ  100
Ok
```

- 2点の座標を読み込んで，距離を出力します．
- 2点の座標を(X1, Y1), (X2, Y2)とすると距離は，

$$距離=\sqrt{|X1-X2|^2+|Y1-Y2|^2}$$

によって得られるので，プログラムでは130行のようになります．

ASC

アスキー：ascii

機能 文字に対応するキャラクタコードを返す

書式 ASC(〈文字列〉)

解説
- 〈文字列〉の最初の文字を，対応するキャラクタコード(数値)に変換します．
- 〈文字列〉にヌルストリングを与えたときには "Illegal function call" エラーが起こります．

例

ASC("9") 返される値 57

ASC("NEC") 返される値 78

ASC(" ") 返される値 "Illegal function call" エラー

プログラム例

```
list⏎
100 ' ASC sample
110 A$=INPUT$(1)
120 IF ASC(A$)=13 THEN END
130 PRINT A$;" ノ キャラクタ コート゛ ハ ";ASC(A$);" テ゛ス。"
140 GOTO 110
Ok
run⏎
a ノ キャラクタ コート゛ ハ  97  テ゛ス。
A ノ キャラクタ コート゛ ハ  65  テ゛ス。
g ノ キャラクタ コート゛ ハ  103  テ゛ス。
i ノ キャラクタ コート゛ ハ  105  テ゛ス。
e ノ キャラクタ コート゛ ハ  101  テ゛ス。
F ノ キャラクタ コート゛ ハ  70  テ゛ス。
o ノ キャラクタ コート゛ ハ  111  テ゛ス。
8 ノ キャラクタ コート゛ ハ  56  テ゛ス。
* ノ キャラクタ コート゛ ハ  42  テ゛ス。
Ok
```

- キーボードから1文字読み込み，そのキャラクタコードを出力します．
- 110行で1文字読み込みます．実行例では "a", "A", "g", "i", "e", "F", "o", "8", "*" そして ⏎ キーを入力しています． ⏎ キーを押すと120行でプログラムが終了します(⏎ キーのキャラクタコードは13です)．

参照：CHR$，資料8　キャラクタコード

ATN

アーク・タンジェント：arc tangent

機　能　逆正接(アークタンジェント)を返す

書　式　ATN(〈数式〉)

解　説

• 〈数式〉の値の逆正接(アークタンジェント)を返します.

• 返される値の単位はラジアン(π/180×角度)で，返される値の範囲は－π/2からπ/2までです.

• ATN関数の演算は単精度で行われます.

例

ATN(1) 返される値 .785398

ATN(0) 返される値 0

プログラム例

```
list
100 ' ATN sample
110 INPUT "value(-1 to 1)";A:IF ABS(A)>1 THEN 110
120 IF ABS(A)=1 THEN AS=1.5708:AC=0:GOTO 150
130 AS=ATN(A/SQR(-A*A+1))
140 AC=-ATN(A/SQR(-A*A+1))+3.14159/2
150 PRINT "arcCOS(";A;")=";AC
160 PRINT "arcSIN(";A;")=";AS
170 END
Ok
run
value(-1 to 1)? .5
arcCOS( .5 )= 1.0472
arcSIN( .5 )= .523599
Ok
```

• －1から1までの数値を読み込んで，逆余弦，逆正弦の値を求めます.

• 130行と140行では，

$$\arcsin A = \arctan\,(A/\sqrt{1-A^2})$$

$$\arccos A = -\arctan\,(A/\sqrt{1-A^2}) + \frac{\pi}{2}$$

を使って計算しています.

参照：SIN, COS, TAN

ATTR$

ディスク

アトリビュート・ダラー：attribute $

機能 ファイルまたはフロッピィディスクの属性を返す

書式 ATTR$(〈ドライブ番号〉)

ATTR$(#〈ファイル番号〉)

ATTR$(〈ファイル名〉)

解説 • 指定したファイル，フロッピィディスクの属性を返す関数で，与えられるパラメータによって次の値を返します．

〈ドライブ番号〉……指定したドライブに入っているフロッピィディスクの属性

〈ファイル番号〉……ファイル番号によってオープンされているファイルの属性

〈ファイル名〉………ファイル名で指定されたファイルの属性

• 属性は，3文字の文字列で返され次のような意味を持ちます．

"␣␣␣"：属性は"なし"です．読み込みおよび書き込みが可能な状態です．

"R␣␣"：書き込みのときにリードアフターライト(書き込んだ内容とメモリ上の内容の比較チェック)を行います．

"␣␣P"：書き込みが禁止されています．

• 属性を表す文字列の2文字目は常に空白で意味を持ちません．

例 ATTR$(1) 返される値 (ドライブ1に入っているフロッピィディスクの属性)

ATTR$(#2) 返される値 (ファイル番号2でオープンしているファイルの属性)

ATTR$("demo . nsr") 返される値 ("demo . nsr"というファイル名がついたファイルの属性)

プログラム例

ディスク

```
list
100 ' ATTR$ sample
110 INPUT "ﾌｧｲﾙ ﾒｲ : ";F$
120 AT$=ATTR$(F$)
130 PRINT "ﾌｧｲﾙ ";F$;" ﾉ ｿﾞｸｾｲ ﾊ : ";
140 A$="ｶｲｼﾞｮ"
150 IF AT$="R  " THEN A$="ﾘｰﾄﾞ ｱﾌﾀｰ ﾗｲﾄ"
160 IF AT$="  P" THEN A$="ﾗｲﾄ ﾌﾟﾛﾃｸﾄ"
170 PRINT A$+" ﾃﾞｽ｡"
```

```
180 END
Ok
run↵
ﾌｧｲﾙ ﾒｲ : ? data↵
ﾌｧｲﾙ data ﾉ ｿﾞｸｾｲ ﾊ : ｶｲｼﾞｮ ﾃﾞｽ｡
Ok
```

• 指定されたファイルの属性を出力します.

• 120行で，指定されたファイルの属性を表す文字を変数AT$に代入しています.

参照：SET

AUTO

オート：auto

機　能　行番号を自動的に発生させる

書　式　AUTO〔〈行番号〉〕〔,〈増分〉〕

解　説
- AUTOコマンドは，プログラム入力時に行番号を自動的に発生させるモードを指定します．
- 〈行番号〉は，最初に発生させる行番号で，以後，[↵]キーの入力ごとに〈増分〉を加えた行番号を発生させます．
- 〈行番号〉は0～65529まで，〈増分〉は1～65529までの整数であり，指定のないときは，それぞれ10とみなされます．
- [STOP] キーを押すと，AUTOモードから抜け出し，BASICのコマンドレベルに戻ります．このとき，最後に発生させた行番号の行は格納されません．
- AUTOモードでは，現在編集中の1行以外の行をスクリーンエディタによって編集することはできません．AUTOモードを抜け出してから修正してください．

例

AUTO
- 10行から10行刻みで行番号を発生させるモードに入ります．

AUTO 1000,10
- 1000行から10行刻みで行番号を発生させるモードに入ります．

実 行 例

```
auto 100,20[↵]
100 ' AUTO sample[↵]
120 print "auto test"[↵]
140 end[↵]
160 [STOP]
Ok
list[↵]
100 ' AUTO sample
120 PRINT "auto test"
140 END
Ok
```

- 行番号を自動的に発生させてプログラムを入力します．
- この実行例では，100行から20行刻みに行番号を発生させています．
- 140行まで入力したら，160行が発生した時点で [STOP] キーを押してコマンドレベルに戻ります．

BEEP

ビープ：beep

機能　ブザーを鳴らす

書式　BEEP〔〈スイッチ〉〕

解説
- 内蔵スピーカによりブザーを鳴らしたり，止めたりします．
- 〈スイッチ〉が1のときブザーは鳴り続け，〈スイッチ〉が0で止まります．
- 〈スイッチ〉を省略した場合には，一定時間ブザーを鳴らします．

これは，PRINT CHR$(7); を実行するか，CTRL＋Gを入力したのと同じです．

例　BEEP
- 一定時間ブザーを鳴らします．

BEEP 0
- ブザーを止めます．

プログラム例

```
list
100 ' BEEP sample
110 FOR I=&H41 TO &H44
120   PRINT CHR$(I);" ヲ ニュウリョク シテ クダ゛サイ。 ";
130   A$=INPUT$(1)
140   IF A$=CHR$(I) THEN PRINT A$ ELSE BEEP:GOTO 130
150 NEXT I
160 END
Ok
run
A ヲ ニュウリョク シテ クダ゛サイ。 A
B ヲ ニュウリョク シテ クダ゛サイ。 B
C ヲ ニュウリョク シテ クダ゛サイ。 C
D ヲ ニュウリョク シテ クダ゛サイ。 D
Ok
```

- 入力する文字を間違えたときブザーを鳴らします．
- "A"から"D"までの文字を順番に入力するようにメッセージを表示しますので，指定された文字を入力してください．違う文字を入力した場合はブザーが鳴り，正しい文字を入力するまで待ち続けます．
- 140行で入力した文字と指定した文字が同じであれば，その文字を出力し，同じでなければブザーが鳴り，130行の入力待ち状態に戻ります．

参照：資料7　コントロールコード表

BGM

ビー・ジー・エム：bgm

機　能　並列動作モードの制御を行う

書　式　BGM〈スイッチ〉

解　説

- 並列動作を行うか行わないかを決めます．
- 〈スイッチ〉の値が1で並列動作モードになり，0で並列動作モードが解除されます．
- 並列動作モードは，PLAY文などで音を出しながら次の命令を実行しますので，音を中断させることなく実行することができます．
- このモードを解除した状態では，音を出している間，次の命令の実行を待ちます．
- システム起動時には並列動作モードになっています．

例　BGM 1

- 音楽機能を並列動作モードにします．

プログラム例

```
list⏎
100 ' BGM sample
110 STOPM
120 BGM 0
130 FOR I=0 TO 61
140   PRINT "ﾈｲﾛ ﾊﾞﾝｺﾞｳ =";I
150   PLAY "@=I;CEG"
160 NEXT I
170 END
Ok
```

- 現在演奏中の音色番号を表示しながら音を出します．
- 120行で並列動作モードを解除しているため，150行のPLAY文が終わるまでは次の音色番号が表示されません．

BLOAD

ディスク

ビー・ロード：binary load

機能 フロッピィディスクから機械語プログラムをロードする

書式 BLOAD〈ファイル名〉〔,ロードアドレス〉〕〔,R〕

解説

• 〈ファイル名〉によって指定されたフロッピィディスク上の機械語プログラムファイルをメモリにロードします．このファイルは，BSAVEコマンドで作成されたものでなければなりません．

• 〈ロードアドレス〉が省略された場合，BSAVEコマンドで指定した番地からロードされます．また，〈ロードアドレス〉が指定されるとその番地からロードされます．

• Rオプションを指定すると，プログラムロード後，セーブの際に指定された〈セーブアドレス〉から，プログラムを実行します．このとき既に開かれているファイルはその状態を保持します．〈ロードアドレス〉が指定されている場合は，〈ロードアドレス〉からプログラムを実行します．

例 BLOAD "demo.bin",&HD000

• "demo. bin"というファイル名の機械語ファイルを，D000H番地からロードします．

実　行　例

ディスク

```
clear 512,&hdfff↵
Ok
bload "2:test.m",&he000↵
Ok
```

• &he000番地から機械語プログラムをロードします．

• ドライブ2に入っているフロッピィディスクの中の"test. m"というファイルを，&he000番地からロードします．BLOADコマンドの前のCLEAR文で機械語プログラム領域を確保しています．

参照：BSAVE

BSAVE

ディスク

ビー・セーブ：binary save

機能 機械語プログラムをフロッピィディスクにセーブする

書式 BSAVE〈ファイル名〉,〈セーブアドレス〉,〈長さ〉

解説
- メモリ上に置かれている機械語プログラムを指定された〈ファイル名〉でフロッピィディスクへセーブします.
- 〈セーブアドレス〉で指定された番地から,〈長さ〉バイトの内容を機械語プログラムとしてセーブします.
- 〈セーブアドレス〉を実行開始番地としておくと,BLOADコマンドにより,機械語プログラムのロードと同時に実行を開始するこのができます.

例 BSAVE "demo.bin",&HE000,500
- E000H番地から500バイトの機械語プログラムを,"demo. bin"というファイル名でセーブします.

実　行　例

ディスク

```
bsave "2:test.m",&he000,&h100
Ok
```

- &he000番地から&he0ff番地までの内容を,"test. m"というファイル名でドライブ2に入っているフロッピィディスクにセーブします.

参照：BLOAD

CDBL

コンバート・ダブル：convert to double

機能 整数値，単精度実数値を倍精度実数値に変換した値を返す

書式 CDBL（〈数式〉）

解説 •〈数式〉の値を倍精度実数値に変換します．ただし，型変換が行われるだけで有効桁数の変化はありません．

•結果の値の精度は，変換する前の型と同じ(整数型なら整数部のみ，単精度実数型なら有効数字 6 桁)になります．

C

例 CDBL(−3) 返される値 −3

CDBL(1.23) 返される値 1.230000019073486

プログラム例

```
list
100 ' CDBL sample
110 A=1.23456:GOSUB 150
120 A=123456!:GOSUB 150
130 A=1.23456E+08:GOSUB 150
140 END
150 A#=CDBL(A)
160 PRINT A,A#
170 RETURN
Ok
run
 1.23456        1.234559893608093
 123456         123456
 1.23456E+08    123456000
Ok
```

• 3つの単精度実数を倍精度実数に変換して出力します．

•150行目からのサブルーチンで，変数Aに与えられている単精度実数を倍精度実数に変換して表示します．

参照：CINT，CSNG

CHR$

キャラクタ・ダラー：character $

機能 キャラクタコードに対応する文字を返す

書式 CHR$(〈数式〉)

解説
- 〈数式〉の値をキャラクタコードとし，対応する文字，またはコントロールコードを返します．
- 〈数式〉は，0～255の値とします．もし，この範囲にない場合は，"Illegal function call"エラーとなります．

例

CHR$(&H41) 返される値 "A"

CHR$(12) 返される値 画面消去コード

プログラム例

```
list
100 ' CHR$ sample
110 INPUT "number(0-255)";A
120 IF A<0 OR 255<A THEN END
130 PRINT "ｷｬﾗｸﾀ ｺｰﾄﾞ";A;" ﾉ ﾓｼﾞ ﾊ ";CHR$(A);" ﾃﾞｽ｡"
140 GOTO 110
Ok
run
number(0-255)? 57
ｷｬﾗｸﾀ ｺｰﾄﾞ 57  ﾉ ﾓｼﾞ ﾊ 9 ﾃﾞｽ｡
number(0-255)? 65
ｷｬﾗｸﾀ ｺｰﾄﾞ 65  ﾉ ﾓｼﾞ ﾊ A ﾃﾞｽ｡
number(0-255)? 34
ｷｬﾗｸﾀ ｺｰﾄﾞ 34  ﾉ ﾓｼﾞ ﾊ " ﾃﾞｽ｡
number(0-255)? 42
ｷｬﾗｸﾀ ｺｰﾄﾞ 42  ﾉ ﾓｼﾞ ﾊ * ﾃﾞｽ｡
number(0-255)? -1
Ok
```

- キャラクタコードを読み込んで，それに対応する文字を出力します．
- 120行で，0から255以外の値を読み込んだ場合プログラムを終了します．
- 130行でCHR$関数を使って文字を出力しています．

参照：ASC，資料8　キャラクタコード

CINT

コンバート・インテジャー：convert to integer

機能 単精度実数値，倍精度実数値を整数値に変換する

書式 CINT（〈数式〉）

解説

- 〈数式〉の値の小数部分を切り捨てて整数型に変換した値を返します．
- 結果の値が－32768～32767の範囲にないときには，"Overflow" エラーが起こります．
- この関数は，INT関数のようにただ数値を整数化するだけでなく，型の変換も行います．

例

CINT(1.2345) 返される値 ⇒ 1

CINT(−3.456) 返される値 ⇒ −4

CINT(56789) 返される値 ⇒ "Overflow" エラー

プログラム例

```
list↵
100 ' CINT sample
110 INPUT "A,D";A,D%
120 GOSUB 140
130 END
140 AD=CINT(A*10^D%+.5)/10^D%
150 PRINT A,AD
160 RETURN
Ok
run↵
A,D? 1.2345,3↵
 1.2345          1.235
Ok
```

- 小数点以下を任意の桁数で四捨五入します．
- 変数Aは値，D%は桁数を表します．
- 140行では，まずAに$10^{D\%}$をかけて，必要な桁数を整数部に持っていきます．それに0.5を加えCINT関数によって小数点以下を切り捨てることにより四捨五入します．最後に$10^{D\%}$で割って小数部であった部分をもとに戻しています．

参照：CSNG，CDBL，INT

グラフィックステートメント

CIRCLE

サークル：circle

機能　円(楕円)を描く

書式　CIRCLE ｜(gx , gy) ｜ STEP(sx , sy)｜ , 〈半径〉〔 , 〈パレット番号〉〕〔 , 〈開始角度〉〕〔 , 〈終了角度〉〕〔 , 〈比率〉〕

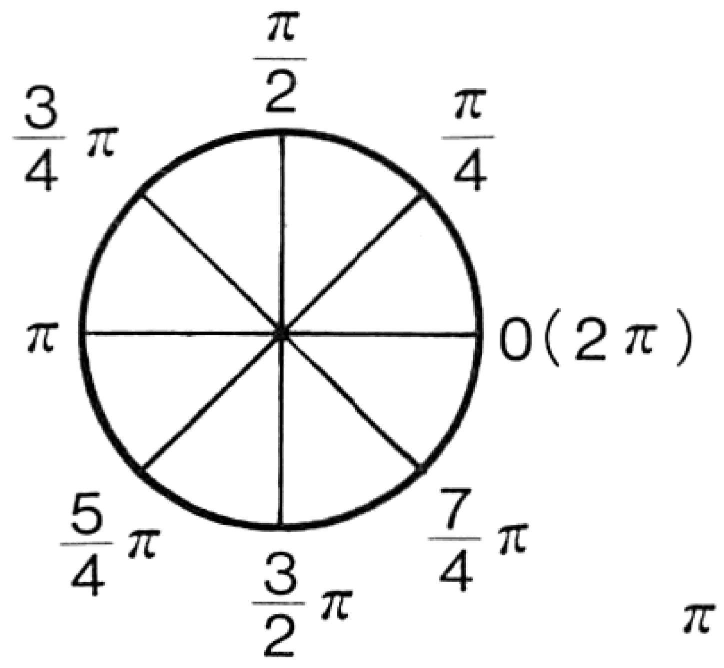
解説

• グラフィック座標(gx, gy)を中心とし，〈半径〉で指定される大きさの円を描きます．〈半径〉が負のときは，"Illegal function call"エラーとなります．また〈半径〉に16384以上の値を使用すると正しく動作しない場合があります．

• 〈パレット番号〉が指定されると，指定されたパレットの色で円を描きます．省略された場合には，COLOR文で指定されている〈フォアグラウンドカラー〉で円を描きます．

• 〈開始角度〉，〈終了角度〉が指定されると，指定された角度の範囲内に円弧を描きます．角度の指定の単位はラジアン(π/180×角度)で，省略値はそれぞれ0と2πです．〈開始角度〉，〈終了角度〉は，−2π〜2πの範囲で指定します．範囲を超えた場合"Overflow"エラーとなります．

• 〈開始角度〉，〈終了角度〉が負であった場合には，その絶対値をとり，正にした角度が用いられますが，そのとき中心から半径が描かれますので扇形を描くことができます．



π＝3.1415

• 〈比率〉は，(垂直方向の半径)/(水平方向の半径)で指定します．省略された場合には，640×200ドットのモードでは0.5，320×200ドットのモードでは1が用いられます．

• 〈比率〉が1以下の場合には，〈半径〉は水平方向の半径を意味します．また1以上の場合には，〈半径〉は垂直方向の半径を意味します．

• CIRCLE文を実行すると，LPは円の中心座標(gx, gy)に設定されます．

例　CIRCLE(100,100),50

- グラフィック座標(100, 100)を中心に，半径50の円を描きます．

プログラム例

```
list
100 ' CIRCLE sample
110 SCREEN 0,0:CLS 3
120 DATA 25,5,40,13,17
130 FOR I=1 TO 5
140   READ DAT:EN=ST+DAT/100*3.1415*2
150   CIRCLE(320,100),150,I,-ST,-EN
160   ST=EN
170 NEXT I
180 END
Ok
```

- 円グラフを描きます．
- 変数ST，ENは，それぞれ開始角度，終了角度を表します．
- 120行のデータを比率として円グラフを描きます．

参照：COLOR

CLEAR

クリア：clear

機　能　変数の初期化およびメモリ領域の設定を行う

書　式　CLEAR〔〈ストリング領域の大きさ〉〔,〈メモリの上限〉〕〕

解　説

・すべての数値変数を0に，文字変数を""(ヌルストリング)に初期化します．また，配列宣言も無効となります．

・〈ストリング領域の大きさ〉はBASICが文字列を処理するために用いるメモリ領域の大きさをバイト数で指定します．多くの文字列演算を行ったり大きな文字配列を用いる場合には，この領域を大きく設定してください．リセット後の初期化された状態では，この領域は512に設定されています．

・〈メモリの上限〉はBASICが使用するメモリの上限番地を指定します．その番地以後に置かれたデータや機械語プログラムは，BASICによって破壊されることはありません．

・CLEAR文が実行される以前に，DEF文(DEF FN文，DEF USR文，DEF INT文)によって定義あるいは指定された情報は，CLEAR文の実行によってすべて無効となります．

例

CLEAR

・変数を初期化します．

CLEAR 2000，&HCFFF

・変数を初期化し，文字列領域を2000バイトとし，さらに，BASICで使用するメモリの上限をCFFFH番地とします．

プログラム例

```
list↵
100 ' CLEAR sample
110 A%=213:B=74.7543:C#=76346.43263654303#
120 GOSUB 160
130 CLEAR
140 GOSUB 160
150 END
160 PRINT "A%=";A%,"B!=";B!,"C#=";C#
170 PRINT:PRINT
180 RETURN
Ok
run↵
A%= 213        B!= 74.7543    C#= 76346.43263654303


A%= 0          B!= 0          C#= 0


Ok
```

• CLEAR文を実行する前と実行した後の変数の値を出力します.
• 130行で数値変数を0に初期化しています.

参照：FRE，3.5　機械語プログラムの呼び方

CLOAD

シー・ロード：cassette load

機　能　カセットテープからプログラムを読み込む

書　式　CLOAD〈ファイル名〉

解　説　• CLOADコマンドは，カセットテープから指定した〈ファイル名〉のプログラムを読み込む命令です．

• 〈ファイル名〉は文字列で指定し，先頭の6文字が有効となります．

• CLOADコマンドの後，カセットテープから指定した〈ファイル名〉を見つけると

Found :〈ファイル名〉

と表示し，プログラムをロードします．そのとき，画面右上にアスタリスク(*)が点滅し，ロード中であることを示します．

• もし，指定した〈ファイル名〉と違った〈ファイル名〉を見つけると

Skip :〈ファイル名〉

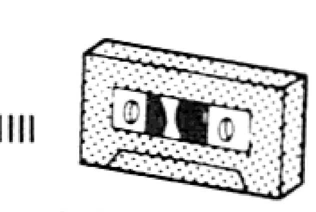
と表示して，目的の〈ファイル名〉を探し続けます．

• プログラムを正常に読み込むと画面に"Ok"と表示します．

例　CLOAD "demo"

• "demo"というファイル名のプログラムをロードします．

実　行　例


カセットテープ

```
cload "sample"↵
Found:sample
Ok
```

• カセットテープから"sample"というファイル名のプログラムをロードします．

参照：CLOAD?, CSAVE

CLOAD?

シー・ロード・クエスチョン：cassette load ?

機能 メモリ上のプログラムとテープ上のプログラムとを比較する

書式 CLOAD?〈ファイル名〉

解説 • CLOAD? コマンドは，現在メモリ上にあるプログラムと〈ファイル名〉で指定されたテープ上のプログラムを比較(ベリファイ)します.

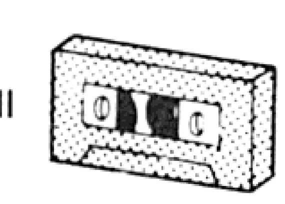
• すべての内容が等しければ"Ok"と表示し，そうでなければ"Bad"と表示してからコマンドレベルに戻ります.

• CLOAD? コマンドは，メモリ上のプログラムがミスなくカセットテープ上にセーブされているかを確認するためのものであり，CSAVE コマンドに引き続いて実行します.

例 CLOAD? "demo"

• "demo" というファイル名でセーブしたプログラムの比較確認を行います.

実　行　例



カセットテープ

```
csave "test"↵
Ok
cload? "test"↵
Found:test
Ok
```

• 現在メモリ上にあるプログラムを"test"という名前でカセットテープにセーブします．その後，カセットテープを巻き戻してからカセットテープ上とメモリ上のプログラムを比較します.

注意：• CLOAD? コマンドはCSAVEコマンドの直後に行ってください.

• CSAVEコマンドのあと1度プログラムを実行すると，CLOAD? コマンドでプログラムの比較を行っても正しく実行できないことがあります.

参照：CSAVE, CLOAD

CLOSE

ディスク

クローズ：close

機能 ディスクファイルを閉じる

書式 CLOSE〔〔#〕〈ファイル番号〉〔,〔#〕〈ファイル番号〉〕..〕

解説

• 〈ファイル番号〉に対応するファイルを閉じます．以後，CLOSE文によって指定された〈ファイル番号〉は，異なるファイルを開くために再び利用することができます．また，閉じられたファイルは，同じあるいは異なったファイル番号によって再び開くことができます．

• CLOSE文では，〈ファイル番号〉を複数指定することにより一度に複数のファイルを閉じることができます．〈ファイル番号〉が省略された場合には，そのとき開いているファイルをすべて閉じます．

• 閉じているファイルに対して，データの入出力を行うことはできません．

• ファイルが出力用にオープンされていた場合には，必ずCLOSE文を実行しなければなりません．

• Rオプション指定のないRUNコマンド，END文，NEWコマンドを実行すると自動的にすべてのファイルを閉じます．STOP文または，STOPキーでプログラムの実行を中止した場合，ファイルを閉じる作業は行いませんので注意してください．

例

CLOSE

• オープンされているすべてのファイルを閉じます．

CLOSE #2

• ファイル番号2のファイルを閉じます．

プログラム例

ディスク

```
list
100 ' CLOSE sample
110 OPEN "2:data4" FOR OUTPUT AS #1
120 PRINT #1,"file data 1"
130 CLOSE #1
140 OPEN "2:data5" FOR OUTPUT AS #1
150 PRINT #1,"file data 2"
160 CLOSE #1
170 END
Ok
```

• ドライブ2に入っているフロッピィディスクに"data 4","data 5"というファイルを作ります．

注意：• ファイルを開いたままフロッピィディスクを取り出すと，フロッピィディスクの配置表（FAT）を壊す恐れがあります．したがって，ファイルをオープンした場合，必ずファイルを閉じてからフロッピィディスクを取り出してください．

参照：OPEN

CLS

シー・エル・エス：clear screen

機能 画面を消去する

書式 CLS〔〈機能〉〕

解説 •〈機能〉は，1，2，3の値をとり，それぞれ次のように働きます．
省略されたときは1が選択されます．

1：テキスト画面を消去します．

2：グラフィック画面のビューポート内を消去します．グラフィック画面がカラーモードのときは，COLOR文で指定した〈バックグラウンドカラー〉で消去します．

3：テキスト画面とグラフィック画面の両方を消去します．

•テキスト画面の消去を指定したとき，COLOR文で〈ヌルキャラクタコード〉が指定されている場合は，その文字でテキスト画面を塗りつぶします．

•グラフィック画面の消去を指定したとき，消去されるのはそのときにアクティブなページだけです．

•グラフィック画面を消去したとき，LPはビューポートの左上の頂点に移動します．また，テキスト画面を消去したとき，カーソルはホームポジション(画面の左上)に移動します．

例 CLS 2

•グラフィック画面のビューポート内を消去します．

プログラム例

```
list↵
100 ' CLS sample
110 SCREEN 0,0
120 GOSUB 180:BEEP:CLS 1
130 GOSUB 180:BEEP:CLS 2
140 GOSUB 180:BEEP:CLS 3
150 GOSUB 180:VIEW(200,50)-(400,150)
160 BEEP:CLS 3
170 END
180 FOR I=1 TO 20
190   FOR J=1 TO 50:PRINT "*";:NEXT J
200   LINE(0,0)-(I*30,199),7
210 NEXT I
220 RETURN
Ok
```

•テキスト画面に"＊"を表示し，グラフィック画面に線を引いて，CLS 1, CLS 2, CLS 3を実行します．

• 150行でビューポートを設定しているため，160行のCLS 3では，グラフィック画面はビューポートの中だけ消去されます．

参照：SCREEN, COLOR, VIEW

COLOR

カラー：color

機　能　テキスト画面とグラフィック画面の色の指定を行う

書　式　COLOR〔〈ファンクションコード〉〕〔,〈バックグラウンドカラー〉〕〔,〈ヌルキャラクタコード〉〕〔,〈フォアグラウンドカラー〉〕

解　説

•〈ファンクションコード〉は，テキスト画面に表示する文字の色などを設定します．この機能は，テキスト画面がカラーモードのときと白黒モードのときとで働きが異なります．

白黒モードの場合（CONSOLE，，，0）

0－ノーマル（通常の表示）
1－シークレット（文字を表示しない）
2－ブリンク（点滅する）
3－シークレット（1と同じ）
4－リバース（反転する）
5－リバースシークレット（反転して文字は表示されない）
6－リバースブリンク（反転して点滅する）
7－リバースブリンクシークレット（5と同じ）

カラーモードの場合（CONSOLE，，，1）

0－黒，　1－青，　2－赤，　3－紫
4－緑，　5－水色，　6－黄色，　7－白

•〈バックグラウンドカラー〉とは，グラフィック画面の地の色のことです．グラフィック画面がカラーモードのときは，CLS文によってグラフィック画面を消去するときこの色で塗りつぶされ，パレット番号で指定します．

白黒モードのときは，COLOR文の実行後，地の色が〈バックグラウンドカラー〉となります．このときの色は，パレット番号ではなくカラーコードで指定します．

•〈ヌルキャラクタコード〉は，テキスト画面を消去するとき使用する文字のキャラクタコードを指定します．通常は32（20H）を指定します．

•〈フォアグラウンドカラー〉とは，グラフィック画面に点や線を描くときに使われるパレット番号のことです．種々のグラフィックステートメントでパレット番号を指定しなかった場合，この〈フォアグラウンドカラー〉が用いられます．

例

```
COLOR,0,,7
```

• グラフィック画面のフォアグラウンドカラーを7(パレット番号7)，バックグラウンドカラーを0(パレット番号0)に設定します．

プログラム例

```
list
100 ' COLOR sample
110 SCREEN 0,0:CONSOLE ,,,1
120 COLOR 7,1,,7:CLS 3
130 GOSUB 180
140 FOR I=1 TO 7
150   COLOR I,,,I:GOSUB 180
160 NEXT I
170 END
180 PRINT "COLOR"
190 LINE(100,50)-STEP(200,100),,BF
200 RETURN
Ok
```

• 黒以外の7色で，四角形を描きます．

• 120行で，テキスト画面の文字を白，バックグラウンドカラーを青，フォアグラウンドカラーを白に設定しています．次のCLS 3が実行されると，グラフィック画面が青になります．

• 180行の"COLOR"という文字は，直前に実行されたCOLOR文の〈ファンクションコード〉で指定された色で出力されます．

• 190行のLINE文では，直前に実行されたCOLOR文の〈フォアグラウンドカラー〉で指定された色で，四角形を描きます．

参照：CONSOLE, SCREEN, CLS，資料8　キャラクタコード

COLOR=

カラー・イコール：color ＝

機能 カラーパレットを変更する

書式 COLOR＝(〈パレット番号〉,〈カラーコード〉)

解説 ・グラフィック画面への色の指定には，カラーパレットを用います．この命令は，どのパレットにどの色を対応させるかを決めるものです．

・〈パレット番号〉とは，0から7までの8つのカラーパレットにつけられた固有の番号で，それぞれのパレットにどの色を持ってくるかは〈カラーコード〉によって指定します．

・〈パレット番号〉と〈カラーコード〉の関係は次のとおりです．

〈パレット番号〉		〈カラーコード〉
0		0（黒）
1		1（青）
2	どのカラーパレットに	2（赤）
3	どのカラーコードを	3（紫）
4	対応させるかを任意に	4（緑）
5	決めることができる	5（水色）
6		6（黄色）
7		7（白）

・システム起動時には，カラーパレットはそのパレット番号と同じカラーコードが割り付けられています．したがって，もしカラーパレットを全く変更しなければ，カラーパレットとカラーコードは同じものとして使ってかまいません．

・カラーパレットの変更後は，グラフィック画面に表示されている色，およびこれから使うすべてのグラフィック命令(PSET文，LINE文など)におけるカラー表示は，そのパレット番号に割り付けられているカラーコードに設定されます．

・パレットは，グラフィックに対してのみ有効です．したがって，テキスト画面のカラーはパレットの変更に影響されません．

例 COLOR＝(1,4)

・カラーパレット1に，カラーコード4（緑）を割り付けます．

プログラム例

```
list↵
100 ' COLOR= sample
110 SCREEN 0,0:CLS 3:COLOR=(7,7)
```

```
120 CIRCLE(320,100),50,7
130 PAINT(320,100),7,7
140 FOR J=1 TO 100
150   FOR I=1 TO 7
160     COLOR=(7,I)
170     FOR K=0 TO 100:NEXT K
180   NEXT I
190 NEXT J
200 END
Ok
```

• グラフィック画面の中央に白く塗った円を描き，色を次々と変化させます．

• 120行と130行で白い円を描いて，その中を白く塗りつぶします．

• 150行から180行で，COLOR＝文を使ってパレット番号7の色を，カラーコード1から7の色に次々と変えていきます．これによって円の色が次々と変化します．

参照：COLOR

COLOR@

カラー・アットマーク：color @

機能 テキスト画面に書かれた文字に色や機能を設定する

書式 COLOR@(cx1，cy1)－(cx2，cy2)，〈ファンクションコード〉

解説 • テキスト画面のキャラクタ座標の2点(cx1，cy1)，(cx2，cy2)を対角とする四角形の領域に書かれている文字に色を付けたり，ブリンクなどの機能を設定したりします．

• 指定する座標は，キャラクタ座標で次の範囲となります．

cx1，cx2 ＝ { 0～(〈桁数〉－1)　白黒モード
　　　　　　　0～(〈桁数〉－2)　カラーモード

cy1，cy2＝0～(〈行数〉－1)

〈桁数〉，〈行数〉はWIDTH文で指定された値です．

• 〈ファンクションコード〉は，CONSOLE文によって設定されているモードによって持つ意味が異なります．この機能及び指定の仕方はCOLOR文の〈ファンクションコード〉の場合と全く同様ですから，そちらを参照してください．

例 COLOR@(0，0)－(9，9)，4 （カラーモードの場合）

• テキスト画面の左上(10桁×10行)を緑に設定します．

プログラム例

```
list
100 ' COLOR@ sample
110 CONSOLE 0,25,0,1
120 WIDTH 80,25:CLS
130 FOR I=0 TO 15
140   PRINT "COLOR@test"
150 NEXT I
160 FOR I=0 TO 7
170   COLOR@(0,I*2)-(9,I*2+1),I
180 NEXT I
190 END
Ok
```

• "COLOR@test"という文字を8色に分けて出力します．

• 130行から150行で，"COLOR@test"という文字を縦に16行分表示します．

• 160行から180行で，テキスト画面を2行ごとに色を指定します．

注意：• テキスト画面がカラーモードのとき，グラフィック画面の左端に重ならないので，グラフィック画面の全領域に色を付けることはできません．

また，この命令によって設定された領域の上に新しく文字を書いた場合，書かれた文字はこの影響を受けません．

CONSOLE

コンソール：console

機　能　テキスト画面のモード設定を行う

書　式　CONSOLE〔〈スクロール開始行〉〕〔,〈スクロール行数〉〕〔,〈ファンクションキー表示スイッチ〉〕〔,〈カラー／白黒スイッチ〉〕〔,〈文字フォント〉〕

解　説

• テキスト画面に関するさまざまなモード設定を行います.

•〈スクロール開始行〉と〈スクロール行数〉を指定することにより，画面上でのスクロールする領域(スクロールウィンドウ)を指定します.

•〈スクロール開始行〉は，どの行からスクロールを開始するかを指定します.

•〈ファンクションキー表示スイッチ〉に1を指定すると，画面最下行にファンクションキーに登録されている文字列を表示します．0を指定した場合は表示されません.

•〈カラー／白黒スイッチ〉を1にするとテキスト画面をカラーモードにし，0を指定した場合には白黒モードとなります.

•〈文字フォント〉は，テキスト画面に表示される文字フォントを選択します．0を指定すると，標準モード(カタカナモード)に，1を指定するとひらがなモードになります.

• システム起動時は，〈スクロール開始行〉,〈スクロール行数〉,〈ファンクションキー表示スイッチ〉,〈カラー／白黒スイッチ〉,〈文字フォント〉は，それぞれ，0, 20, 1, 0, 0に設定されます.

• それぞれのパラメータを省略したときは，現在定義されている値と同じになります.

例

CONSOLE 0, 10, 1, 1

• 最上行から10行をスクロールウィンドウとし，ファンクションキーを表示するカラーモードに設定します.

CONSOLE , , , ,1

• 文字フォントをひらがなモードとします.

プログラム例

```
list
100 ' CONSOLE sample
110 CLS:CONSOLE 0,25,0,1
120 FOR I=1 TO 25 STEP 4
130   CONSOLE 0,I
140   GOSUB 170
```

```
150 NEXT I
160 END
170 CLS
180 FOR J=1 TO 40
190   PRINT J
200   FOR K=0 TO 10:NEXT K
210 NEXT J
220 RETURN
Ok
```

- スクロール行数を変えて，1から40までの数字を出力します．
- 130行で，スクロール行数を設定しています．

注意：• プリンタにひらがなを出力する場合は，あらかじめプリンタをひらがなモードに設定しておく必要があります．詳しくは，資料8　キャラクタコード「プリンタを標準モード，ひらがなモードに切り替える方法」を参照してください．

参照：資料8　キャラクタコード

CONT

コント：continue

機能 停止したプログラムの実行を再開する

書式 CONT

解説 • CONTコマンドは，STOPキーや，STOP文，END文で停止または終了した文の次の文から実行を開始するコマンドです．

• 通常は，プログラムの誤りを探すときに，STOP文と共に用います．

• 実行を停止した後，ダイレクトモードで変数の値を変更したり，表示した後，CONTコマンドによって実行を再開することができます．

• INPUT文，LINE INPUT文で入力待ちのときに実行を停止した場合，それらの命令から実行が再開されます．

• プログラムを変更したとき，あるいはキーボードから入力を行うINPUT$関数をSTOPキーで止めた後は，CONTコマンドによる実行再開はできません．

例 CONT

• プログラムの実行を再開します．

実行例

```
list↵
100 ' CONT sample
110 I=10
120 PRINT I,SQR(I)
130 STOP
140 I=I+1
150 GOTO 120
Ok
run↵
 10            3.16228
Break in 130
Ok
cont↵
 11            3.31663
Break in 130
Ok
cont↵
 12            3.4641
Break in 130
Ok
```

• 10以上の整数の平方根をSTOP文で止めながら出力します．

• 実行例は，130行がなければ10以上の整数とその平方根を次々と出力するものです．

• 実行されると，10とその平方根を出力してすぐ停止します．STOP文による停止なので，CONT文を入力することによってSTOP文の次の140行から実行を再

開することができます.

参照：STOP, END

COPY

ディスク

コピー：copy

機能　画面に表示されている情報をプリンタに出力する

書式　COPY〔〈機能〉〕

解説

•テキスト画面，グラフィック画面に表示されている文字や図形をプリンタに出力します．

•〈機能〉は，1から5までの値をとり，次のように働きます．省略されたときは，3が選択されます．

1：テキスト画面のみをプリンタにコピーします．

2：グラフィック画面のみをプリンタにコピーします．

3：テキスト画面，グラフィック画面の両方をプリンタにコピーします．

4：グラフィック画面のみをプリンタに，縦方向を2分の1に縮小してコピーします．

5：テキスト画面，グラフィック画面の両方をプリンタに，縦方向を2分の1に縮小してコピーします．

例　COPY 2

•グラフィック画面のみをプリンタに出力します．

プログラム例

ディスク　プリンタ

```
list
100 ' COPY sample
110 SCREEN 0,0:WIDTH 80,25:CLS 3
120 FOR I=1 TO 20
130   X=RND(1)*75:Y=RND(1)*19
140   LOCATE X,Y:PRINT "COPY";
150   CIRCLE(X*8,Y*10),30
160 NEXT I
170 FOR I=1 TO 5
180   COPY I
190 NEXT I
200 END
Ok
```

•テキスト画面のいろいろな位置に"COPY"という文字を出力し，グラフィック画面のいろいろな位置に円を描いてから，画面をプリンタに出力します．

•120行から160行で，画面に"COPY"と円を描きます．

•170行から190行で，COPY 1からCOPY 5まで，順番に実行します．

注意：•COPY文が使用可能なプリンタは次のとおりです．これら以外のプリンタに対してのCOPY文の動作は保証されません．

- 縦横の比率がほぼ画面上と同等にコピーできるもの

  PC-8023, PC-8023C, PC-8024, PC-8027, PC-8824, PC-8825, PC-PR101, PC-PR201およびこれらの同等品

- 縦長(約4/3倍)になるがコピーできるもの

  PC-8821, PC-8822, PC-PR103, PC-PR202, PC-PR403およびこれらの同等品

- グラフィック画面のコピーにおいて，画面に表示されたものと，プリンタにコピーされたものとでは，いくらかイメージが異なります．たとえば，画面上では真円であっても，プリンタに出力した場合に多少縮んだ形になります．

COS

コサイン：cosine

機能 余弦(コサイン)の値を返す

書式 COS(〈数式〉)

解説
- 〈数式〉の値の余弦(コサイン)を返します.
- 〈数式〉の値の単位は，ラジアン(π/180×角度)です.
- COS関数の演算は単精度で行われます.

例

COS (3.14159/180＊30) 返される値 .866026 (COS (30°)の値)

COS (3.14159/180＊60) 返される値 .500001 (COS (60°)の値)

プログラム例

```
list⏎
100 ' COS sample
110 INPUT "ｶｸﾄﾞ (ﾄﾞ) : ";D
120 CS=COS(D/180*3.14159265358979#)
130 PRINT "COS";USING "###";D;:PRINT " = ";CS;
140 PRINT "SEC";USING "###";D;
150 IF ABS(CS)<=1E-06 THEN PRINT " = ﾃｲｷﾞ ｻﾚﾏｾﾝ"
                      ELSE PRINT " = ";1/CS
160 END
Ok
run⏎
ｶｸﾄﾞ (ﾄﾞ) : ? 45⏎
COS 45 =  .707107 SEC 45 =  1.41421
Ok
```

- 角度を度で読み込んで，コサイン(COS)とセカント(SEC)の値を出力するプログラムです.
- セカントの値を求めるには

$$\sec\theta = 1/\cos\ \theta$$

を用いて計算します.
- cosの値が0.000001以下になると，単精度実数ではEのついた指数形式になるので，150行ではそのような場合はオーバーフローとみなすようにしてあります.

参照：SIN, TAN, ATN

CSAVE

シー・セーブ：cassette save

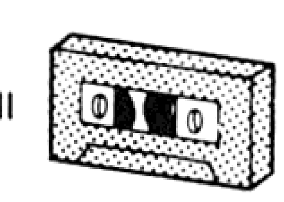
機能 カセットテープへプログラムをセーブする

書式 CSAVE〈ファイル名〉

解説
- CSAVEコマンドはメモリ上のプログラムを指定された〈ファイル名〉でカセットテープにセーブします．
- 〈ファイル名〉は，先頭の6文字が有効となります．
- プログラムのセーブが完了すると，画面に"Ok"と表示してコマンドレベルに戻ります．

例 CSAVE "demo"
- メモリ上のプログラムを"demo"というファイル名でカセットテープにセーブします．

実行例




```
csave "GAME1"↵
Ok
```

- 現在メモリ上にあるプログラムを，"GAME1"というファイル名でカセットテープにセーブします．

注意：• CSAVEコマンドを実行した後は，CLOAD?コマンドを実行してプログラムが正常にセーブできているかを確認してください．

参照：CLOAD, CLOAD?

CSNG

コンバート・シングル：convert to single

機能 整数値，倍精度実数値を単精度実数値に変換した値を返す

書式 CSNG(〈数式〉)

解説
• 〈数式〉の値を有効数字6桁の単精度実数値に変換した値を返します．
• 結果の値が −1.70141E+38〜1.70141E+38 の範囲にないときは，"Overflow" エラーが起こります．

例

CSNG (123) 返される値 ▶ 123

CSNG (3.14159) 返される値 ▶ 3.14159

CSNG (−1E+40) 返される値 ▶ "Overflow" エラー

プログラム例

```
list⏎
100 ' CSNG sample
110 INPUT"A,D";A#,D%
120 GOSUB 140
130 END
140 AD#=INT(A#*10^D%+.5)/10^D%
150 PRINT A#,CSNG(AD#)
160 RETURN
Ok
run⏎
A,D? 1.25,1⏎
 1.25          1.3
Ok
run⏎
A,D? 0.1234567,5⏎
 .1234567      .12346
Ok
```

• 小数点以下を任意の桁数で四捨五入します．
• 140行では，CINT関数のプログラム例と同じことを実行していますが，ここでは，CINT関数でなくINT関数を使っています．
• 150行で変数AD#をそのまま出力すると，誤差を含んだ結果が出てくるので，CSNG関数を使って有効桁数はそのままで単精度実数に変換してから出力しています．

参照：CINT，CDBL

CSRLIN

カーソル・ライン：cursor line

機能 カーソルの垂直位置を返す

書式 CSRLIN

解説
• 現在のカーソルの垂直位置を行単位で返します．
• 得られる値は，25行モードで0から24，20行モードでは0から19となります．なお，画面の最上行が0，最下行が24(または19)となっています．

例 CSRLIN
● 現在のカーソルの垂直位置を返します．

プログラム例

```
list
100 ' CSRLIN sample
110 WIDTH 80,25:CLS
120 F=1:X=39:Y=2
130 GOSUB 150
140 GOTO 130
150 LOCATE X,Y:PRINT " ";
160 IF CSRLIN>=22 OR CSRLIN<=1 THEN F=F*-1
170 Y=Y+F:LOCATE X,Y:PRINT "●";
180 RETURN
Ok
```

• カーソルの位置を調べながら"●"を上下に動かします．
• 160行で，カーソルの位置を調べ，1行目か22行目のときに動く方向を変えます．
• 変数Fは方向を表し，1のときは下，−1のときは上を表します．こうすることによって次のY座標はY＋Fで求めることができます．

参照：POS

CVI/CVS/CVD

ディスク

シー・ブイ・アイ：convert to integer
シー・ブイ・エス：convert to single
シー・ブイ・ディ：convert to double

機能 文字列を数値データに変換した値を返す

書式 CVI(〈2文字の文字列〉)

CVS(〈4文字の文字列〉)

CVD(〈8文字の文字列〉)

解説 •フロッピィディスク上のランダムファイルから読み込んだデータは文字型になっているためCVI, CVS, CVD関数を使って，数値データに変換します．

•CVI関数は2文字の文字列(すなわち2バイトのデータ)を整数値に，CVS関数は4文字の文字列を単精度実数値に，CVD関数は8文字の文字列を倍精度実数値にそれぞれ変換します．

•指定した関数の文字列より長い文字列を代入した場合，代入した文字列の左側から指定されている文字数までが変換されます．たとえば，CVI("ABCD")とした場合，"AB"だけが変換されます．

例 CVI("AB") 返される値 16961

•2バイトの文字コード"AB"を整数値16961(内部表現＝4241H)に変換します．数値の内部表限については，VARPTR関数を参照してください．

プログラム例

ディスク

```
list
100 ' CVI/CVS/CVD sample
110 OPEN "2:rdata" AS #1
120 FIELD #1,2 AS D1$,4 AS D2$,8 AS D3$
130 GET #1,1
140 A%=CVI(D1$)
150 B=CVS(D2$)
160 C#=CVD(D3$)
170 PRINT "A =";A%
180 PRINT "B =";B
190 PRINT "C =";C#
200 CLOSE
210 END
Ok
run
A = 123
B = 8534.13
C = 3.141592654000002
Ok
```

•ランダムファイルからデータを読み取り数値に変換して表示します．

•MKI$/MKS$/MKD$のプログラム例を実行することによって作成された"rdata"というファイルの内容を出力します．

• 140行から160行で，2バイトの文字列を整数型，4バイトの文字列を単精度実数型，8バイトの文字列を倍精度実数型に変換しています．

参照：MKI$/MKS$/MKD$

DATA

データ：data

機能 READ文で読み込まれる数値，文字定数を格納する

書式 DATA〈定数〉〔,〈定数〉...〕

解説 • DATA文は，READ文で読み込まれる数値や文字定数を格納します．

• この命令は，REM文などと同じように非実行文(何の動作も行わない命令)で，プログラムのどこにおいてもかまいません．

• 1つのDATA文には，プログラムの1行(255文字)に入るだけのデータをセットすることができ，また1つのプログラム中にいくつものDATA文を置くことができます．

• READ文は，行番号の小さい方から順番にDATA文の中のデータを読み込んでいきます．

• READ文で読まれる回数よりデータの個数が少ない場合(READ文に対応するデータがない場合)"Out of DATA"エラーとなります．

• 〈定数〉は，数値定数(整数，固定小数点，浮動小数点，8進数，16進数)，文字定数のいずれかです．ただし，定数式(2＋3など)は許されません．また，READ文で読み込まれる場合，READ文で指定されている変数の型は，対応するデータの定数の型と一致しなければなりません．

• DATA文の中のデータは，コンマ(,)で区切られますが，文字定数の中にコンマ，コロン(:)，または文字列の前後に意味のある空白を含むときは，その文字定数全体をダブルクォーテーション(")で囲む必要があります．

• RESTORE文によってREAD文で読み込むDATA文を行単位で指定することができます．

例 DATA 8001, "COMPUTER"

• データとして，8001という数値データ，"COMPUTER"という文字列データを格納します．これらは，READ文によって読み出されます．

プログラム例

```
list
100 ' DATA sample
110 DATA 10,20
120 READ A,A$
130 PRINT A:PRINT A$
140 END
Ok
run
 10
20
Ok
```

D

• DATA文に格納されている定数を読み取り，出力します．

• 110行で，10と20というデータを設定します．

• 120行のREAD文では，AとA$という変数に，10，20というデータをそれぞれ代入します．10は，単精度実数型，20は文字型として読み込まれます．

参照：READ, RESTORE

DATE$

デート・ダラー：date $

機能 内蔵のカレンダ時計の日付を返す

書式 DATE$

解説 • DATE$変数を参照することにより内蔵カレンダ時計の日付が得られます.

• 日付は，8文字の文字列で返され，文字変数に代入したり，PRINT文で内容を表示することができます.

• 日付を設定するには，DATE$変数に次の形式の文字列を代入します.

"YY/MM/DD"

YY ‥年(00～99)
MM‥月(01～12)
DD ‥日(01～31)
} 各2文字で指定します.

例 DATE$ [返される値] (現在の日付を返す)

DATE$= "85/08/19"

• 内蔵カレンダ時計に，1985年8月19日を設定します.

プログラム例

```
list
100 ' DATE$ sample
110 PRINT "ｷｮｳﾊ 19";
120 PRINT LEFT$(DATE$,2);" ﾈﾝ ";
130 PRINT MID$(DATE$,4,2);" ｶﾞﾂ ";
140 PRINT RIGHT$(DATE$,2);" ﾆﾁ ﾃﾞｽ"
150 END
Ok
run
ｷｮｳﾊ 1985 ﾈﾝ 01 ｶﾞﾂ 12 ﾆﾁ ﾃﾞｽ
Ok
```

• 日付を出力します.

注意：• 内蔵のカレンダ時計はバッテリーバックアップにより常に日付を更新しています．ただし，年の値は保持されておらず，システム起動時に，"85"に初期化されます．年の値として"85"以外を使用する際は，システム起動時ごとに再設定を行ってください.

• DISKバージョンを使用すれば，付属のユーティリティを使用することによってこの手間を省くことができます.(詳しくは，N_{80SR}-BASIC (DISKバージョン) ユーザーズガイドを参照してください.)

参照：TIME$

DEF FN

ディファイン・ファンクション：define function

機能 新しく関数を定義する

書式 DEF FN〈名前〉〔(〈パラメータリスト〉)〕=〈関数の定義式〉

解説
- 関数の名前は"FN"+〈名前〉で定義します(例：FNPW).
- 〈名前〉は変数名として正しい形をしたものでなくてはならず最初の2文字のみが区別されます.
- 〈パラメータリスト〉はその関数が呼ばれたときに，〈関数の定義式〉中の同じ変数名に対応します．変数名の間はコンマで区切ります.
- 〈パラメータリスト〉に書かれた変数名は〈関数の定義式〉を評価する際にのみ有効となります．したがってプログラム中に同じ変数名があっても影響はありません.
- 〈関数の定義式〉は，その関数の演算内容を記述する式で，1行の範囲に限られます．このユーザ定義関数は，数値関数，文字関数，その混在のいずれでもかまいませんが，〈パラメータリスト〉内の変数と，それに渡される変数とが同一の型でなければなりません.
- 〈関数の定義式〉中で使われている変数が，〈パラメータリスト〉中にない場合は，その変数が，その時点で持っている値が使われます.
- DEF FN文はダイレクトモードで使用することはできません.

例 DEF FNLG(X)=LOG(X)/LOG(10)

- 常用対数を求める関数として"FNLG"を定義します．以後，プログラム中で，FNLG(〈数式〉)を使うと，〈数式〉の常用対数値を得ることができます.

プログラム例

```
list↵
100 ' DEF FN sample
110 DEF FNPW(X,Y)=X^Y
120 INPUT "x,y";X,Y
130 PRINT "x^y=";FNPW(X,Y)
140 END
Ok
run↵
x,y? 2,8↵
x^y= 256
Ok
```

- 2つの実数x, yを読み込んで，x^yを出力します.
- 110行で引数x, yを持ち，x^yを返す関数FNPWを定義しています.

DEFINT/SNG/DBL/STR

ディファイン・イント/シングル/ダブル/ストリング：define integer/single/double/string

機能 変数の型宣言を行う

書式 DEF | INT | 〈文字の範囲〉〔,〈文字の範囲〉...〕
| SNG |
| DBL |
| STR |

解説

• 〈文字の範囲〉で指定された文字ではじまるすべての変数名の型を，次に示す型宣言により定義します．

DEFINT ：変数を整数型に
DEFSNG：変数を単精度型に
DEFDBL：変数を倍精度型に
DEFSTR：変数を文字型に
} 宣言します．

• 〈文字の範囲〉は，1文字の英字で指定します．また，ある範囲にわたって指定するときは，1文字の英字と1文字の英字をハイフン(-)でつなぎます．ただし，最初の英字は，後の英字よりもアルファベット順で先のものでなければなりません．

• 型宣言文字で指定した場合，これらのDEF宣言よりも優先されます．

• 型宣言が行われていない文字で始まる変数は，すべて単精度実数型変数とみなされます．

例 DEFINT I,J

• IとJで始まる変数を整数として宣言します．以後，プログラム中では整数型として扱われます．たとえば，IXはIX%と等しくなります．

プログラム例

```
list
100 ' DEF INT/SNG/DBL/STR sample
110 DEFINT I-M
120 DEFSNG A,B
130 DEFDBL D
140 DEFSTR S
150 I1=1.23:J=65.643
160 AB=1.23:BB=65.643
170 D=3.141592653589792#
180 D%=3.141592654000001#
190 S=" STRING"
200 PRINT "I1 =";I1,,"J=";J
210 PRINT "AB =";AB,,"BB =";BB
220 PRINT "D =";D,"D% =";D%
230 PRINT "S =";S
```

```
240 END
Ok
run⏎
I1 = 1                              J= 65
AB = 1.23                           BB = 65.643
D = 3.141592653589792               D% = 3
S = STRING
Ok
```

• 整数型，単精度実数型，倍精度実数型の数を，それぞれ出力するプログラムです．

• 110行でI, J, K, L, Mではじまる変数を整数型，120行でA, Bではじまる変数を単精度実数型，130行でDではじまる変数を倍精度実数型，140行でSではじまる変数を文字型にそれぞれ定義します．

• 180行では，Dではじまる変数名に倍精度実数値を代入していますが，型宣言文字%で，整数型に指定されているので，D%という変数は，3という値を持つことになります．

DEF USR

ディファイン・ユーザ：define user

機能 機械語で作られたユーザ関数の実行開始番地を定義する

書式 DEF USR〔〈番号〉〕＝〈開始番地〉

解説
• USR関数(ユーザ関数)が呼び出す機械語プログラムの実行開始番地の設定を行います.

•〈番号〉は0～9までの値で，USR＋〈番号〉が関数名となり，関数名によって機械語で作られたプログラムが呼び出されます.〈番号〉を省略したときは0とみなされます.

•〈開始番地〉は機械語プログラムの実行開始番地を指定します.

• DEF USR文によって定義されていないUSR関数を参照すると，"Illegal function call" エラーになります.

例 DEF USR3 ＝ &HDA00

• USR関数の実行開始番地を，DA00H番地に設定します.

プログラム例

```
list
100 ' DEF USR sample
110 CLEAR 300,&HBFFF
120 DEF USR=&HC000:GOSUB 210
130 CONSOLE 0,25,0,1:WIDTH 80,25:CLS
140 FOR I=1 TO 80*24
150   PRINT "♥";
160 NEXT I
170 CLS:FOR I=1 TO 500:NEXT I
180 VAD%=&HF3C8
190 I=USR(VAD%):FOR I=1 TO 500:NEXT I
200 END
210 FOR AD=&HC000 TO &HC015
220   READ DA:POKE AD,DA
230 NEXT AD
240 RETURN
250 DATA &H7E           :'      LD   A,(HL)
260 DATA &H23           :'      INC  HL
270 DATA &H66           :'      LD   H,(HL)
280 DATA &H6F           :'      LD   L,A
290 DATA &H3E,&HE9      :'      LD   A,"♥"
300 DATA &H11,&H28,&H00 :'      LD   DE,40
310 DATA &H0E,&H19      :'      LD   C,25
320 DATA &H06,&H50      :' L1:  LD   B,80
330 DATA &H77           :' L2:  LD   (HL),A
340 DATA &H23           :'      INC  HL
350 DATA &H10,&HFC      :'      DJNZ L2
360 DATA &H19           :'      ADD  HL,DE
370 DATA &H0D           :'      DEC  C
380 DATA &H20,&HF6      :'      JR   NZ,L1
390 DATA &HC9           :'      RET
Ok
```

• BASICと機械語のプログラムで，テキスト画面全体に"♥"を2回書きます.

・１回目は140行から160行のBASICプログラムで実行しますが，２回目は250行以下のDATA文で定義されている機械語で実行します．

・120行で，機械語の開始番地がC000H番地であることを定義します．

・180行で変数VAD%にテキストVRAMの開始番地を代入し，190行で，それを引数として機械語ルーチンを呼び出します．

・210行から240行までのサブルーチンで，DATA文に定義されている機械語データをメモリに書き込みます．

参照：USR

DELETE

デリート：delete

機　能　プログラムを行単位で削除する

書　式　DELETE〔〈始点行番号〉〕〔−〈終点行番号〉〕

解　説　•〈始点行番号〉から〈終点行番号〉までのプログラム行を削除します．

•〈始点行番号〉だけを指定した場合はその行だけを削除します．この場合，DELETEコマンドを省略して行番号だけを入力したのと同じになります．

•ハイフン(−)と〈終点行番号〉のみを指定した場合，プログラムの先頭から指定行までを削除します．

例　DELETE −990

•プログラムの先頭から990行までを削除します．

実　行　例

```
list↵
100 ' DELETE sample
110 '
120 ' DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ
130 ' ｻｸｼﾞｮ ｼﾏｽ
140 PRINT " DELETE sample"
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT " ｻｸｼﾞｮ ｼﾏｽ"
170 END
Ok
delete 110-130↵
Ok
list↵
100 ' DELETE sample
140 PRINT " DELETE sample"
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT ". ｻｸｼﾞｮ ｼﾏｽ"
170 END
Ok
```

•100行から170行までのプログラムのうち，110行から130行までをDELETEコマンドを使って削除しています．

DIM

ディメンション：dimension

機能　配列変数の要素の大きさを指定する

書式　DIM〈変数名〉(〈添字の最大値〉〔,〈添字の最大値〉...〕)

解説

• 配列を使用する前には，DIM文によって配列宣言を行います．

• DIM文は，配列変数の添字の最大値を設定し，同時にメモリ上にその配列の領域を割り当てるものです．

• 添字の最大値が10以下の場合は，配列宣言をする必要はありません．このとき，配列を使用した時点でメモリ上に，添字の最大値を10として領域を割り当てます．

•〈添字の最大値〉の個数は，その配列変数の次元を示します．また，次元はメモリ容量の許す限り255次元まで使用できます．

•〈添字の最大値〉より大きい値の添字を持つ配列が用いられた場合は，"Subscript out of range"エラーになります．

• DIM文が実行された時点で，その配列のすべての要素の値は0（文字型の場合はヌルストリング）に設定されます．

• 既に，DIM文で配列宣言されている配列変数を再度DIM文で配列宣言すると，"Redimensioned array"エラーになります．このような場合は，ERASE文で，配列宣言を取り消した後，DIM文を実行してください．

例　DIM MT(3,3,3)

• 3×3×3次元の配列変数MTの配列宣言を行います．この場合，添字の最大値が10以下なので，配列宣言の必要はないのですが，メモリの節約のためにDIM文を実行します．

プログラム例

```
list↵
100 ' DIM sample
110 DIM KU(9,9)
120 FOR I=1 TO 9:FOR J=1 TO 9
130   KU(I,J)=I*J
140 NEXT J,I
150 FOR I=1 TO 9:FOR J=1 TO 9
160   PRINT USING "####";KU(I,J);
170 NEXT J:PRINT:NEXT I
180 END
Ok
run↵
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
```

```
   4    8   12   16   20   24   28   32   36
   5   10   15   20   25   30   35   40   45
   6   12   18   24   30   36   42   48   54
   7   14   21   28   35   42   49   56   63
   8   16   24   32   40   48   56   64   72
   9   18   27   36   45   54   63   72   81
Ok
```

- 九九の表を出力します.
- 110行で 9×9の 2 次元配列KUを宣言しています.

参照：ERASE

DSKF

ディスク・エフ：disk file

機能　フロッピィディスクの各種情報を返す

書式　DSKF(〈ドライブ番号〉〔,〈機能〉〕)

解説

・〈ドライブ番号〉で指定されたドライブ，およびそこに入っているフロッピィディスクに関する情報を関数の値として返します．

・〈機能〉は値として返す情報の種類を指定します．〈機能〉として指定する値と，そのときに返される関数の値の意味は次のようになっています．

省略したとき：指定したドライブに入っているフロッピィディスクの残り容量をクラスタ数で返す．

指定したとき：指定したドライブの下記の情報を返す．

0：最大トラック番号(＝片面あたりのトラック数－1)
1：1トラックあたりのセクタ数
2：片面フロッピィディスク(0を返す)あるいは両面フロッピィディスク(1を返す)
3：1トラックあたりのクラスタ数
4：フロッピィディスク1枚あたりのクラスタ数
5：ディレクトリのあるトラック番号
6：1クラスタあたりのセクタ数
7：FATの開始セクタ番号
8：FATの終了セクタ番号
9：FATの数
10：IDのあるセクタ番号

・ドライブの種類やディレクトリ，クラスタ，FATについては，資料5　ディスクファイルの構造を参照してください．

例

DSKF(1)　返される値⇒　(ドライブ1に入っている フロッピィディスクの残り容量)

DSKF(2,0)　返される値⇒　(ドライブ2の最大トラック番号)

・5.25インチ両面倍密度ミニフロッピィディスクのとき39，8インチフロッピィディスクのとき76を返します．

プログラム例

ディスク

```
list
100 ' DSKF sample
110 INPUT "drive number";DR
120 PRINT "Track  =";DSKF(DR,0);"/Drive"
130 PRINT "Sector =";DSKF(DR,1);"/Track"
140 PRINT "Disk space =";DSKF(DR)
150 END
Ok
run
drive number? 2
Track  = 39 /Drive
Sector = 16 /Track
Disk space = 120
Ok
```

• 指定されたドライブの最大トラック数，1トラック当りのセクタ数及びフロッピィディスクの残り容量を出力します．

DSKI$

ディスク
ディスク・アイ・ダラー：disk input $

機能 フロッピィディスクから直接データを読み込む

書式 DSKI$(〈ドライブ番号〉〔,〈サーフェス番号〉〕,〈トラック番号〉,〈セクタ番号〉)

解説 •フロッピィディスク上の指定したセクタから直接データを読み込みます.

•データは0番バッファに読み込まれるとともに，256文字の文字列を値として返します.

•文字列の長さは255文字までしか許されませんから，DSKI$関数の値として256バイトのデータすべてを一度に得ることはできません．そこで，データを得るときは，次の方法を用います.

(1) ランダムファイルの場合と同じようにFIELD文により，0番バッファへ複数の文字変数を割り当てる.

(2) VARPTR関数により，0番バッファの置かれているメモリ番地を調べ，その領域をPEEK関数を使って読み出す.

•指定する〈ドライブ番号〉のフロッピィディスクが5.25インチ片面倍密度ミニフロッピィディスクのときは，〔,〈サーフェス番号〉〕を省略します（コンマを含め省略）.

•〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なります.

これらの情報は，DSKF関数によって得ることができます.

例 DSKI$(1,0,18,1)

•ドライブ1に入っているフロッピィディスクの，サーフェス0，トラック18，セクタ1の内容を，0番バッファに読み込みます.

プログラム例

ディスク

```
list
100 ' DSKI$ sample
110 FIELD #0,128 AS A$(0),128 AS A$(1)
120 INPUT "Drive,Surface,Track,Sector";DR,SU,TR,SE
130 DU$=DSKI$(DR,SU,TR,SE)
140 FOR I=0 TO 1
150   FOR J=0 TO 7
160     FOR K=1 TO 16
170       D$=HEX$(ASC(MID$(A$(I),J*16+K,1)))
180       PRINT RIGHT$("0"+D$,2);" ";
190     NEXT K:PRINT
200 NEXT J,I:END
Ok
```

```
run↵
Drive,Surface,Track,Sector? 1,1,2,4↵
4A F2 11 20 D8 20 4A F1 0F FF 3A 49 F1 49 F4 12
00 0B AA 9E 02 58 32 F1 58 3A 59 32 F1 59 3A 58
F1 53 58 28 49 2C 4A 29 3A 59 F1 53 59 28 49 2C
4A 29 00 4C AA A8 02 8B 20 4A F4 12 F2 11 20 D8
20 8B 20 49 F4 12 F2 11 20 D8 20 4D F1 11 3A 89
20 0E C6 02 20 3A A1 20 4A 4A F1 0F FF 3A 49 49
F1 49 F4 12 20 3A A1 20 4A 4A F1 4A F4 12 3A 49
49 F1 49 00 7B AA B2 02 8B 20 58 F1 53 58 28 49
49 2C 4A 4A 29 20 D8 20 8B 20 59 F0 53 59 28 49
49 2C 4A 4A 29 20 D8 20 4D F1 13 20 3A A1 20 4D
F1 15 00 AA AA BC 02 8B 20 59 F1 53 59 28 49 49
2C 4A 4A 29 20 D8 20 8B 20 58 F0 53 58 28 49 49
2C 4A 4A 29 20 D8 20 4D F1 12 20 3A A1 20 4D F1
14 00 DC AA C6 02 B8 20 B3 28 58 F6 13 2C 59 29
2C 12 2C 14 2C F4 1D 6E 12 03 77 3A B8 20 B3 28
58 32 F6 13 2C 59 32 29 2C 12 2C 13 2C F4 1D 6E
Ok
```

• 指定されたフロッピィディスクのセクタの内容を表示します．このプログラム例は5.25インチ両面倍密度ミニフロッピィディスクのものです．

• 120行でドライブ番号，サーフェス番号，トラック番号，セクタ番号を読み込みます．

• 130行で指定されたセクタの内容を0番のバッファに取り込み，最初の256文字を変数DU$に代入します．しかし文字型の変数は255文字以上は許されないので，最後の1文字は無視されます．

• 170行でバッファの内容を1文字ずつ順に2桁の16進数に変換して変数D$に代入し，180行で出力します．

参照：DSKO$, FIELD, DSKF, VARPTR, PEEK

DSKO$

ディスク

ディスク・オー・ダラー：disk output $

機　能　フロッピィディスクに直接データを書き込む

書　式　DSKO$〈ドライブ番号〉〔,〈サーフェス番号〉〕,〈トラック番号〉,〈セクタ番号〉

解　説

• フロッピィディスク上の指定したセクタへ直接データを書き込みます.

• DSKO$文は，そのパラメータにより指定したセクタに0番バッファの内容(256バイト)を書き込みます.

• 書き込むデータは，次の方法で準備します.

(1) ランダムファイルの場合と同じようにFIELD文により，0番バッファに変数領域を割り当てた後，LSET文，RSET文により行う.

(2) VARPTR関数により，0番バッファの置かれているメモリ番地を調べ，POKE文によりデータを書き込む.

• 指定する〈ドライブ番号〉のフロッピィディスクが5.25インチ片面倍密度ミニフロッピィディスクのときは，〔,〈サーフェス番号〉〕を省略します(コンマを含め省略).

• 〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なります.

これらの情報は，DSKF関数によって得ることができます.

例　DSKO$ 1,0,18,2

• ドライブ1に入っているフロッピィディスクの，サーフェス0，トラック18，セクタ2に，0番バッファの内容(256バイト)を書き込みます.

プログラム例

ディスク

```
list↵
100 ' DSKO$ sample
110 FIELD #0,1 AS A$,1 AS B$,254 AS C$
120 LSET A$=CHR$(0)
130 INPUT "Drive";DR:IF DR<0 OR 1<DR THEN 130
140 INPUT "How many files";H$
150 H=VAL(H$)
160 IF H<0 OR 15<H THEN 140
170 LSET B$=CHR$(H)
180 LINE INPUT "Messege? ";M$
190 LSET C$=M$
200 IF DSKF(DR,2)=0 THEN DSKO$ DR,18,13
                    ELSE DSKO$ DR,1,18,13
210 END
Ok
run↵
Drive? 2↵
How many files? 4↵
Messege? width 80,25↵
Ok
```

• フロッピィディスクのIDセクタを書き換えます．このプログラム例は5.25インチ片面倍密度ミニフロッピィディスクと5.25インチ両面倍密度ミニフロッピィディスクの場合です．

• IDセクタは，5.25インチ片面倍密度ミニフロッピィディスクの場合トラック18，セクタ13で，5.25インチ両面倍密度ミニフロッピィディスクの場合サーフェス1，トラック18，セクタ13ですから，200行のようにしてIDセクタに書き込みます．

注意：• この命令は現在のディスクファイルを壊す恐れがありますので，フロッピィディスクおよびファイルの構成を正しく理解した上で用いてください．

参照：DSKI$, FIELD, DSKF

END

エンド：end

機能 プログラムを終了する

書式 END

解説
- プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルに戻ります．
- END文は，プログラムの実行を終了させたいところに置けばよく，また，いくつ置いてもかまいません．
- プログラムの最後のEND文は省略することができます．ただし，この場合，ファイルのクローズは行われません．

例 END

- プログラムを終了する．

プログラム例

```
list↵
100 ' END sample
110 PRINT "Hit 'e' key to end!"
120 A$=INPUT$(1)
130 IF A$="e" THEN END ELSE PRINT A$:GOTO 120
Ok
run↵
Hit 'e' key to end!
g
h
i
w
u
Ok
```

- "e"が入力されるまで，入力した文字を出力します．
- 130行で，プログラムを終了させるか，入力された文字を出力して次の入力を待つかを判断しています．
- 実行例では"g"，"h"，"i"，"w"，"u"そして"e"を入力しています．

注意：• カセットテープにセーブするプログラムには最後にEND文を付ける習慣をつけてください．END文が付いていないプログラムをロードした後実行した場合，正常に動作しないことがあります．

E

EOF

ディスク

エンド・オブ・ファイル：end of file

機能 ファイルが終了したかどうかを値で返す

書式 EOF(〈ファイル番号〉)

解説 • シーケンシャルファイル(ディスクファイル)において，〈ファイル番号〉で指定されたファイルが終わりに達したかどうかを調べる関数です.

• ファイルが終わりに達したとき，EOF関数は真(−1)，そうでなければ偽(0)を返します.

• ファイル番号で指定されたファイルは，入力モード(INPUTモード)でオープンされていなければなりません.

例 EOF(1) 返される値 −1(ファイルが終わりに達している)

0(まだ終わりではない)

• ファイル番号1のファイルが終わりに達していれば"−1"，そうでなければ"0"の値を返します.

プログラム例

ディスク

```
list
100 ' EOF sample
110 OPEN "tdata" FOR OUTPUT AS #1
120 FOR I=1 TO 10
130   PRINT #1,I
140 NEXT I
150 CLOSE
160 OPEN "tdata" FOR INPUT AS #1
170 IF EOF(1) THEN 200
180 INPUT #1,A:PRINT A
190 GOTO 170
200 END
Ok
run
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
Ok
```

• "tdata"というシーケンシャルファイルを作った後，その内容を読み出して画面に出力します.

• 110行から150行で，1から10までの数をファイルに書き込みます.

• 170行で，ファイルを最後まで読み込んだかどうかを判断しています．

ERASE

イレーズ：erase

機能 配列変数の宣言を取り消す

書式 ERASE〈配列名〉〔,〈配列名〉...〕

解説 • 指定した配列変数の宣言を取り消します．以後，同一変数名の配列変数をDIM文で宣言することができます．

• ERASE文は，配列変数が専有していたメモリ領域を解放します．不要になった配列変数を消すことによって，より効率よくメモリを使うことができます．

• DIM文で配列宣言を行っていない配列変数に対してERASE文を実行すると"Illegal function call"エラーとなります．

例 ERASE A,B$

• 配列変数AとB$の配列宣言を取り消します．

プログラム例

```
list
100 ' ERASE sample
110 DIM A(100),B(100)
120 PRINT FRE(0):ERASE A:PRINT FRE(0)
130 DIM A(200):PRINT FRE(0)
140 DIM B(200) :'ｾﾝｹﾞﾝ ﾃﾞｷﾏｾﾝ !
150 END
Ok
run
 6174
 6586
 5774
Redimensioned array in 140
Ok
```

• 一度宣言した配列をERASE文で取り消し，別の大きさで再度宣言を行います．

• 110行で配列A，Bを宣言しています．

• 120行で，配列Aを消去する前と消去した後のメモリの未使用領域の大きさを出力します．

• 130行で配列Aを宣言し直して，メモリの未使用領域の大きさを出力します．

• 140行で配列Bを宣言し直そうとしていますが，Bは消去されていないためエラーが発生します．

• 実行例の数値は，システム構成によって異なります．

参照：DIM

ERL/ERR

エラー・ライン/エラー・ナンバ：error line/error number

機能 エラーが起こったときのエラーの発生した行番号とエラーコードを返す

書式 ERL

ERR

解説 ・エラーが発生した時点で，ERL変数はエラーが発生した行番号を，ERR変数はそのエラーコードを持っています．この値を参照することによって，エラーの起こった行番号と，エラーの種類を知ることができます．

・エラーがダイレクトモードで生じた場合は，ERL変数は値として，65535を返します．

・通常，ERL変数およびERR変数はON ERROR GOTO文によって指定した処理ルーチンにおいて，処理の流れを制御するために使われます．

例 ERL 返される値 ⇒（エラーが発生した行の行番号）

ERR 返される値 ⇒（エラーの種類を示すエラーコード）

プログラム例

```
list↵
100 ' ERL/ERR sample
110 ON ERROR GOTO 160
120 INPUT "x,y";X,Y
130 AN=X^Y
140 PRINT X;"ﾉ";Y;"ｼﾞｮｳ ﾊ";ANS;"ﾃﾞｽ｡"
150 END
160 'error trap
170 IF ERR=11 AND ERL=130 THEN PRINT " ??? ﾃｲｷﾞ ｻﾚﾏｾﾝ｡":END
180 ON ERROR GOTO 0
Ok
run↵
x,y? 2,3↵
 2 ﾉ 3 ｼﾞｮｳ ﾊ 8 ﾃﾞｽ｡
Ok
run↵
x,y? -1,3↵
-1 ﾉ 3 ｼﾞｮｳ ﾊ-1 ﾃﾞｽ｡
Ok
run↵
x,y? 0,-1↵
 ??? ﾃｲｷﾞ ｻﾚﾏｾﾝ｡
Ok
```

・２つの実数x，yを読み込んでx^yを出力します．

・160行以降のエラー処理ルーチンにおいて，130行でエラーコード11，"Division by zero" エラーが生じたとき，"??? テイギ サレマセン。" を出力して終わり

ます．その他のエラーのときは180行を実行して通常のエラー処理に入ります．

参照：ON ERROR GOTO

ERROR

エラー：error

機能 エラーの発生をシミュレートする

書式 ERROR〈エラー番号〉

解説

• 〈エラー番号〉の値は1～255までの数値で指定します.

• 指定した値がBASICでエラーコードとして使われている場合は，ERROR文が実行されると，〈エラー番号〉に対応するエラーメッセージを表示し，コマンドレベルに戻ります．また，指定した〈エラー番号〉に対応するエラーメッセージがないときは，"Unprintable error"を表示します.

• ON ERROR GOTO文がある場合，ERROR文が実行されると，エラー処理ルーチンへ分岐します．このとき，ERR変数は指定した〈エラー番号〉の値を持ちます.

例 ERROR 5

• "Illegal function call"エラーを起こします.

プログラム例

```
list
100 ' ERROR sample
110 ON ERROR GOTO 180
120 INPUT "3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ:";N
130 IF FIX(N)<>N THEN ERROR 251
140 IF N<100 OR N>999 THEN ERROR 250
150 PRINT USING "<###>";N
160 ON ERROR GOTO 0
170 END
180 IF ERR=250 THEN PRINT " 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!"
190 IF ERR=251 THEN PRINT " ｼﾞｯｽｳ ﾊ ﾆｭｳﾘｮｸ ﾃﾞｷﾏｾﾝ "
200 PRINT
210 RESUME 120
Ok
run
3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ:? 452369
 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ:? 12.5
 ｼﾞｯｽｳ ﾊ ﾆｭｳﾘｮｸ ﾃﾞｷﾏｾﾝ

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ:? 159
<159>
Ok
```

• 入力された数値のチェックを行い，誤りに対してエラーメッセージを表示します.

• もし実数が入力されたら，130行で251番のエラーを発生させます.

• 100から999の間にない数を入力したら，140行で250番のエラーを発生させます.

- 160行ではプログラムを終了する前に，通常のエラー処理に戻しています．

参照：ERL/ERR，資料10　エラーメッセージ

EXP

イクスポネンシャル：exponential

機　能　e(2.7128)を底とする指数関数の値を返す

書　式　EXP(〈数式〉)

解　説
- 〈数式〉の値である指数関数の値を返します．ただし，〈数式〉の値が87.33655より大きい場合には"Overflow"エラーとなります．
- EXP関数の演算は単精度で行われます．

例

EXP(3) 返される値 ⇒ 20.0855

EXP(−12) 返される値 ⇒ 6.14422E−06

EXP(100) 返される値 ⇒ "Overflow" エラー

プログラム例

```
list
100 ' EXP sample
110 INPUT D
120 HS=(EXP(D)-EXP(-D))/2:HC=(EXP(D)+EXP(-D))/2
130 PRINT "SINH";USING "###";D;:PRINT " = ";HS;
140 PRINT "COSH";USING "###";D;:PRINT " = ";HC;
150 END
Ok
run
? 2
SINH  2 =  3.62686 COSH  2 =  3.7622
Ok
```

- EXP関数を使ってハイパボリック・サイン(SINH)とハイパボリック・コサイン(COSH)の値を求めます．
- $\sinh\theta = (e^{\theta} - e^{-\theta})/2$

  $\cosh\theta = (e^{\theta} + e^{-\theta})/2$

  で求めています．

FIELD

ディスク

フィールド：field

機　能　ランダムファイルのバッファに変数領域を割り当てる

書　式　FIELD　〔#〕〈ファイル番号〉，〈フィールド幅〉AS〈文字変数〉〔，〈フィールド幅〉AS〈文字変数〉．．．〕

解　説

- ファイルバッファをランダムファイルバッファとして使用するために，ファイルバッファの中に文字変数の領域を割り当てます．
- FIELD文はランダムファイルの入出力を行う前に実行します．
- 〈ファイル番号〉はOPEN文によってファイルをオープンしたときに指定した番号です．また，特に〈ファイル番号〉が0のときDSKI$関数，DSKO$文において使用されるランダムバッファを定義します．
- 〈フィールド幅〉は〈文字変数〉に割り当てる文字数（バイト数）です．バッファの大きさは256バイトなので，各文字変数の〈フィールド幅〉の合計は256バイト以内でなければなりません．
- 1つのバッファに対して，何回かに分けてFIELD文を実行することができます．2回目以降FIELD文を使用する場合，以前のFIELD文で指定したフィールド幅の合計をダミーとして他の変数に割り当て，そのあとに追加するフィールドを指定してください．
- FIELD文で指定した〈文字変数〉はLSET/RSET文で定義します．また，この変数をINPUT文，LET文の左辺で使用することはできません．もし使用すると，その変数は，一般の文字変数領域に登録され，FIELD文でのバッファの割り当てが無効となり，正しいファイルの入出力が行われなくなります．
- FIELD文で指定した変数は，FIELD文を実行した時点で，〈フィールド幅〉で指定した文字数のヌルキャラクタ（キャラクタコードが0のキャラクタ）がセットされます．

例　FIELD #1，128 AS A$，128 AS B$

- ファイル番号1のファイルバッファに，文字変数A$，B$の領域を各128バイトずつ割り当てます．

プログラム例

ディスク

```
list
100 ' FIELD sample
110 OPEN "2:data1" AS #1
120 FIELD #1,15 AS A$,15 AS B$,15 AS C$
130 LSET A$="NEC"
```

```
140 LSET B$="Personal"
150 LSET C$="Computer"
160 PUT #1,1
170 GET #1,1
180 PRINT A$;B$;C$
190 END
Ok
run⏎
NEC            Personal       Computer
Ok
```

• "data1"というランダムファイルを作ります.

• 120行で始めの15バイトをA$, 次の15バイトをB$, 次の15バイトをC$にそれぞれ割り当てます.

参照：OPEN, GET, PUT, LSET/RSET

FILES/LFILES

ディスク

ファイルズ/エル・ファイルズ：files/list files

機能 フロッピィディスクに入っているファイルの情報を出力する

書式 (1) FILES〔〈ドライブ番号〉〕

(2) LFILES〔〈ドライブ番号〉〕

解説 •〈ドライブ番号〉で指定されたフロッピィディスク上に登録されているファイルの名前，種類およびその大きさを出力します．

• 書式(1)の場合ディスプレイ画面に，(2)の場合プリンタに出力します．

•〈ドライブ番号〉が省略されたときは，ドライブ1が選択されます．

• ファイルの種類は，出力される〈ファイル名〉と〈拡張子〉との区切り記号が空白の場合には，そのファイルはアスキーセーブされたプログラムファイル，またはデータファイルであることを示します．区切り文字が，ピリオドの場合には，バイナリセーブによって作られたプログラムファイル，アスタリスクの場合にはBSAVEコマンドによって作られた機械語プログラムファイルであることを表します．

• ファイルの大きさは，クラスタを単位として表示されます． 5.25インチミニフロッピィディスクの1クラスタは8セクタ(2048バイト)，8インチフロッピィディスクの1クラスタは26セクタ(6656バイト)です．

例 FILES 2

• ドライブ2に入っているファイルの情報を画面に表示します．

実行例

ディスク

```
files ↵
demo  .nsr 2   demo  *bin 2   donau .nsr 3   mail  .nsr 1   mail  dat 3
maze  .nsr 3   cross .nsr 1   plane .nsr 2   telphn.nsr 4   tel001 dat 2
star  .nsr 1   star  *bin 3   star2 *bin 3
Ok
```

• ドライブ1に入っているフロッピィディスクの内容を表示します．

参照：SAVE, BSAVE

FIX

フィックス：fix

機能 数値の整数部の値を返す

書式 FIX(〈数式〉)

解説 •〈数式〉の絶対値の小数点以下を切り捨て，〈数式〉の符号を付けた値を返します．

•FIX関数とINT関数の値は〈数式〉の値が正のとき等しくなりますが，負のときは値が異なります．（例：INT（−3.5）＝ −4, FIX（−3.5）＝ −3）

例 FIX(1.23) 返される値 1

FIX(−3.2) 返される値 −3

プログラム例

```
list
100 ' FIX sample
110 FOR A=-2 TO 2 STEP .5
120 PRINT "A =";A,"FIX(A) =";FIX(A),"INT(A) =";INT(A)
130 NEXT A
140 END
Ok
run
A =-2          FIX(A) =-2     INT(A) =-2
A =-1.5        FIX(A) =-1     INT(A) =-2
A =-1          FIX(A) =-1     INT(A) =-1
A =-.5         FIX(A) = 0     INT(A) =-1
A = 0          FIX(A) = 0     INT(A) = 0
A = .5         FIX(A) = 0     INT(A) = 0
A = 1          FIX(A) = 1     INT(A) = 1
A = 1.5        FIX(A) = 1     INT(A) = 1
A = 2          FIX(A) = 2     INT(A) = 2
Ok
```

•いくつかの実数に対して，FIX関数とINT関数を実行した結果を出力します．

•FIX関数は小数点以下を切り捨てます．

•INT関数は，たとえばA＝−1.5のとき，−1.5を超えない最大の整数になりますからINT(−1.5)＝−2になります．

参照：INT

FOR...TO...STEP～NEXT

フォー・トゥー・ステップ・ネクスト：for...to...step～next

機能 一連の命令を指定回数だけ繰り返し実行する

書式 FOR〈変数〉＝〈初期値〉TO〈終値〉〔STEP〈増分〉〕

～

NEXT〔〈変数〉〔,〈変数〉...〕〕

解説 • FOR文とNEXT文ではさまれた一連の命令を指定した回数だけ繰り返し実行します．回数の指定は，〈初期値〉，〈終値〉，〈増分〉で決まります．

• 〈変数〉には整数変数，単精度変数を使用します．文字変数，倍精度変数は使用できません．

• 〈初期値〉，〈終値〉，〈増分〉は数式です．

• FOR～NEXTの動作は次のようになります．

(1) 〈変数〉に〈初期値〉が設定されます．

(2) FOR以後からそれに対応するNEXTまで実行されます．

(3) STEPによって指定された〈増分〉を〈変数〉に加え，それを新しい〈変数〉とします．

(4) 〈変数〉の値が〈終値〉と比べられ，〈終値〉に達していなければ(2)へ移り，同じ処理が繰り返されます．

これをFOR～NEXTループと呼びます．

• STEPが省略された場合〈増分〉は1とみなされます．また〈増分〉は負の値をとることもできます．このとき〈初期値〉は，〈終値〉より大きくしなければなりません．この場合〈変数〉が〈終値〉よりも小さくなるまでFOR～NEXTループが繰り返されます．

• 次の場合FOR～NEXTループは1回だけ実行されます．

(1) 〈増分〉が正の値で，〈初期値〉が〈終値〉より大きい場合．

(2) 〈増分〉が負の値で，〈初期値〉が〈終値〉より小さい場合．

(3) 〈初期値〉と〈終値〉が同じ値で〈増分〉が0の場合．

ただし，〈変数〉には〈初期値〉が代入されています．

• FOR～NEXTループは入れ子構造にすることができます．これは，1つのFOR～NEXTループ中にもう1つのFOR～NEXTループを置くことができるということです．このとき，それぞれの〈変数〉は別のものを使います．

・１つのFORに対して，入れ子構造さえ正しければ，NEXTはいくつあってもかまいません．ただし，メモリ容量によって制限されます．メモリ容量が不足した場合は，"Out of memory"エラーになります．

例　FOR I＝1 TO 10:PRINT "＊";:NEXT I

・PRINT "＊";を10回実行します．すなわち，"＊"を10個表示します．

プログラム例

```
list↵
100 ' FOR...TO...STEP~NEXT sample
110 FOR I=1 TO 4
120   PRINT "STEP";I
130   FOR J=1 TO 15 STEP I
140     PRINT J,
150   NEXT J
160   PRINT
170 NEXT I
180 END
Ok
run↵
STEP 1
 1            2            3            4            5
 6            7            8            9            10
 11           12           13           14           15

STEP 2
 1            3            5            7            9
 11           13           15
STEP 3
 1            4            7            10           13

STEP 4
 1            5            9            13
Ok
```

・１から15までの数値を増分を＋1，＋2，＋3，＋4として出力します．

・120行から160行までが４回繰り返し実行され，さらにその中で140行を繰り返し実行しています．

FPOS

ディスク

エフ・ポジション：file position

機能 ファイル中での物理的な位置を返す

書式 FPOS（〈ファイル番号〉）

解説 ・〈ファイル番号〉によって指定されたファイルが最後に読み書きしたフロッピィディスク上の物理的な位置を返します．この値は，トラック0，サーフェス0，セクタ1を0としたときの通し番号です．

・5.25インチ両面倍密度ミニフロッピィディスクの場合，通し番号は次のように割り当てていきます．

サーフェス	0	0	0	…	0	0	1	1	1	…	1	1	0	0	0	…
トラック	0	0	0	…	0	0	0	0	0	…	0	0	1	1	1	…
セクタ	1	2	3	…	15	16	1	2	3	…	15	16	1	2	3	…
通し番号	0	1	2	…	14	15	16	17	18	…	30	31	32	33	34	…

例 FPOS (2)

・ファイル番号2のファイル中での現在位置を返します．たとえば，物理的な位置が，トラック2，サーフェス0，セクタ4（5.25インチ両面倍密度ミニフロッピィディスクの場合）のときは，67（＝32×2（トラック）＋16×0（サーフェス）＋4（セクタ）－1）を返します．

プログラム例

ディスク

```
list↵
100 ' FPOS sample
110 OPEN "2:fdata" FOR OUTPUT AS #1
120 PRINT FPOS(1):PRINT
130 FOR I=1 TO 5
140   PRINT #1,STRING$(126,"1")
150   PRINT #1,STRING$(126,"1")
160   PRINT FPOS(1)
170 NEXT I
180 END
Ok
run↵
 664

 666
 667
 668
 669
 670
Ok
```

・ファイルにデータを書き込みながら，フロッピィディスク上の書き込んだ位置を出力します．

・120行で，ファイルのオープンで読み書きしたフロッピィディスクの位置を出力します．

・140行，150行で256バイトのデータ，つまり1セクタ分のデータを書き込んでいます．

・160行で，フロッピィディスク上のデータを書き込んだ位置を出力します．

参照：LOC

FRE

フリー：free

機能 メモリの未使用領域の大きさを返す

書式 FRE（〈機能〉）

解説
- 〈機能〉が数値型であれば，未使用のプログラムおよび変数領域の大きさ，また，文字型であれば未使用の文字領域の大きさをバイト数で返します．
- 〈機能〉の値は型だけが意味を持ち，値そのものは何でもかまいません．

例
FRE (0) 返される値⇒（未使用のプログラムおよび変数領域の大きさ）

FRE (" ") 返される値⇒（未使用の文字領域の大きさ）

プログラム例

```
list↵
100 ' FRE sample
110 CLEAR 1000,&HD000
120 PRINT "ﾌﾟﾛｸﾞﾗﾑ ｵﾖﾋﾞ ﾍﾝｽｳ ﾉ ﾐｼﾖｳ ﾘｮｳｲｷ ﾊ";FRE(0);"ﾊﾞｲﾄ ﾃﾞｽ"
130 PRINT "ﾓｼﾞﾚﾂ ﾉ ﾐｼﾖｳ ﾘｮｳｲｷ ﾊ";FRE("a");"ﾊﾞｲﾄ ﾃﾞｽ"
140 END
Ok
run↵
ﾌﾟﾛｸﾞﾗﾑ ｵﾖﾋﾞ ﾍﾝｽｳ ﾉ ﾐｼﾖｳ ﾘｮｳｲｷ ﾊ 10370 ﾊﾞｲﾄ ﾃﾞｽ
ﾓｼﾞﾚﾂ ﾉ ﾐｼﾖｳ ﾘｮｳｲｷ ﾊ 1000 ﾊﾞｲﾄ ﾃﾞｽ
Ok
```

- CLEAR文を実行してからプログラムと文字列の未使用領域の大きさを出力します．
- 実行例の数値は，システム構成によって異なります．

参照：CLEAR

GET#

ディスク

ゲット・シャープ：get #

機　能　ランダムファイルのデータをバッファに読み込む

書　式　GET〔#〕〈ファイル番号〉〔,〈レコード番号〉〕

解　説

• 〈ファイル番号〉で指定されたファイル中のデータを，対応するバッファの中に読み込みます．指定されたファイルはランダムアクセスのモードでオープンされていなければなりません．

• 〈レコード番号〉はファイルのレコード番号で，指定されたレコード(256バイト)がバッファに読み込まれます．レコード番号が省略された場合には，直前に行われたGET#文，PUT#文で指定されたレコードの次のレコードが読み込まれます．

• GET#文で読み込んだデータを参照するには，あらかじめFIELD文で割り当てられた変数を使います．

例　GET #3,5

• ファイル番号3でオープンしたランダムファイルの5番目のレコードをバッファに読み込みます．

プログラム例

ディスク

```
list
100 ' GET# sample
110 OPEN "2:address" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 FOR I=1 TO LOF(1)
140   GET #1,I
150   PRINT "NAME",A$
160   PRINT "TEL",B$
170   PRINT "ADDRESS",C$
180 NEXT I
190 CLOSE:END
Ok
run
NAME          ｱｲｶﾜ ﾌｻｺ
TEL           115-787-9898
ADDRESS       ｶﾅｶﾞﾜｹﾝ ｶﾜｻｷｼ ﾄｶｲ
NAME          ﾂﾁﾔ ﾋﾃﾞﾐ
TEL           114-123-6565
ADDRESS       ﾄｳｷｮｳﾄ ｼﾅｶﾞﾜｸ ｲﾅｶ
Ok
```

• PUT#文のプログラム例で作ったデータファイルを読み込んで画面に出力します．

• 130行から180行の間で，すべてのデータをGET#文で読み込み出力します．

G

注意：・GET#文を実行した後INPUT#文，LINE INPUT#文等で同一の〈ファイル番号〉を指定すると，そのバッファに入っているデータから入力が行われますので注意してください．

参照：OPEN, FIELD, PUT#

GET@

ゲット・アットマーク：get @

機能 グラフィック画面のグラフィックパターンを配列に読み込む

書式 GET@ |(gx1 , gy1) / STEP(sx1 , sy1)| — |(gx2 , gy2) / STEP(sx2 , sy2)|
,〈配列変数名〉〔(〈要素ナンバ〉)〕

解説

• グラフィック座標系の2点(gx1 , gy1)と(gx2 , gy2)を対角とする長方形の領域を〈配列変数名〉で指定された配列変数に読み込みます.

• 〈配列変数名〉は，グラフィックパターンを読み込む数値型の配列変数の名前です．また，この配列変数に対して〈要素ナンバ〉を指定することもできます．この〈要素ナンバ〉は，グラフィックパターンを配列変数に読み込む際に，どの要素から格納し始めるかを指定するものです．省略した場合は，配列の最初(〈要素ナンバ〉は0)から格納します.

• GET@文を実行する前に，この配列変数に必要な大きさをDIM文で確保しておきます．この配列変数の大きさは，読み込む領域の大きさ，画面モード，配列変数の型によって異なります.

• 1つの配列に1つのパターンしか読み込まない場合は，次の計算式で求めることができます．複数のパターンの場合は，それぞれのパターンについての大きさを合計し，その分確保しなければなりません.

〈必要なバイト数〉=((〈横のドット数〉+7)¥8)＊〈縦のドット数〉＊M+4

白黒モードのとき………M=1
カラーモードのとき……M=3

〈添字の値〉=〈必要なバイト数〉¥N+1

整数型配列のとき………N=2
単精度型配列のとき……N=4
倍精度型配列のとき……N=8

• GET@文は，PUT@文と密接な関係にあり，通常ペアで使用されます.

• GET@文の実行後，LPは(gx2, gy2)に移動します.

例

```
GET@(0,0)-(15,15),G%
```

• グラフィック座標(0, 0)～(15, 15)の範囲にあるグラフィックパターンを配列変数G%に読み込みます.

プログラム例

```
list⏎
100 ' GET@ sample
110 SCREEN 0,0:CLS 3
120 XD=40:YD=20
130 BYTE=((XD+7)¥8)*YD*3+4
140 FACT=BYTE¥2+1
150 DIM G%(FACT)
160 CIRCLE(XD/2-1,YD/2-1),YD/2,6
170 PAINT(XD/2-1,YD/2-1),6,6
180 GET@(0,0)-STEP(XD-1,YD-1),G%
190 FOR X=0 TO 500 STEP 100
200   PUT@(X,100),G%:PUT@(X,100),G%
210 NEXT X
220 END
Ok
```

- 画面左上に黄色の円を描き，それと同じ円が左から右に移動します．
- 130行，140行で配列宣言で必要な添字の数を計算しています．
- 180行で160行，170行で描いた円を配列に取り込みます．
- 190行から210行で，取り込んだ円を6ヶ所に表示します．
- 2回PUT@文を実行していますが，この場合機能がXORなので，2回目を実行すると画面がもとの状態に戻ります．つまり，一度描いた円を消去することになります．

注意：• 配列変数は数値型の一次元配列でなければなりません．

参照：PUT@

GOSUB

ゴー サブ：go to subroutine

機能 サブルーチンへ制御を移す

書式 GOSUB〈行番号〉

解説 •〈行番号〉から始まるサブルーチンプログラムに制御を移します．つまり，指定した〈行番号〉を呼び出し，その行から実行を続けます．

•サブルーチンで，RETURN文を実行すると，サブルーチンを呼び出したGOSUB文の次の文に制御が戻ります．

•１つのサブルーチンの中から他のサブルーチンを呼び出すこともでき，これをサブルーチンの多重化(ネスティング)と呼びます．この多重化はメモリのスタック領域の容量が許す限り行うことができます．もし，スタック領域が足りなくなった場合には"Out of memory"エラーとなります．

•GOSUB文で呼び出されるサブルーチンは，RETURN文で制御を戻します．

例 GOSUB 1000

•1000行から始まるサブルーチンを呼び出します．

プログラム例

```
list↵
100 ' GOSUB sample
110 INPUT "ﾃｲﾍﾝ  :";BA
120 INPUT "ﾀｶｻ   :";H
130 GOSUB 160
140 PRINT "ﾒﾝｾｷ ﾊ";AR
150 END
160 AR=BA*H/2
170 RETURN
Ok
run↵
ﾃｲﾍﾝ  :? 78↵
ﾀｶｻ   :? 8↵
ﾒﾝｾｷ ﾊ 312
Ok
```

•三角形の底辺と高さを読み込んで，面積を出力します．

•160行と170行が，面積を計算するサブルーチンです．

参照：RETURN

GOTO/GO TO

ゴートゥー：go to

機能 指定した行へ制御を移す

書式 GOTO〈行番号〉またはGO TO〈行番号〉

解説 •〈行番号〉で指定した行へ制御を移します．つまり，〈行番号〉で指定した行から実行を続けます．

•"GO"と"TO"の間に1文字のスペースを置くことができます．

例 GOTO 1000

•1000行へ制御を移します．

プログラム例

```
list↵
100 ' GOTO sample
110 PRINT "How do you do?"
120 GOTO 110
Ok
run↵
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
^C
Break in 110
Ok
```

• STOP キーを押すまで"How do you do?"を続けて出力します．

•実行例では8回ループしたところで STOP キーが押され，110行でプログラムが終了しています．

HEX$

ヘキサ・ダラー：hexadecimal $

機　能　10進数を16進数の文字列に変換した値を返す

書　式　HEX$(〈数式〉)

解　説　•〈数式〉の値を16進数に変換して，その文字列を返します．

•〈数式〉の値の範囲は，－32768～65535です．〈数式〉に小数点が含まれる場合は，小数点以下を切り捨てたあと変換します．

例

HEX$(－32768) 返される値 "8000"

HEX$(5000) 返される値 "1388"

HEX$(65535) 返される値 "FFFF"

プログラム例

```
list↵
100 ' HEX$ sample
110 PRINT "     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9"
120 FOR I=0 TO 9
130   PRINT STR$(I);"- ";
140   FOR J=0 TO 9
150     PRINT RIGHT$("0"+HEX$(I*10+J),2);" ";
160   NEXT J
170   PRINT
180 NEXT I
190 END
Ok
run↵
     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9
 0-  00 01 02 03 04 05 06 07 08 09
 1-  0A 0B 0C 0D 0E 0F 10 11 12 13
 2-  14 15 16 17 18 19 1A 1B 1C 1D
 3-  1E 1F 20 21 22 23 24 25 26 27
 4-  28 29 2A 2B 2C 2D 2E 2F 30 31
 5-  32 33 34 35 36 37 38 39 3A 3B
 6-  3C 3D 3E 3F 40 41 42 43 44 45
 7-  46 47 48 49 4A 4B 4C 4D 4E 4F
 8-  50 51 52 53 54 55 56 57 58 59
 9-  5A 5B 5C 5D 5E 5F 60 61 62 63
Ok
```

• 0から99までの10進数を，16進数に変換した表を出力します．

• 150行では16進数の値が1桁の場合，上位の桁に0を付けて2桁で表示させています．

参照：OCT$

H

IF...THEN...ELSE/ IF...GOTO...ELSE

イフ・ゼン・エルス/イフ・ゴートゥー・エルス：if...then...else/if...goto...else

機能 論理式の条件判断を行い次に実行する文を選択する

書式

```
IF〈論理式〉|THEN |  〈文〉  || |〔ELSE|  〈文〉  |〕
            |     |〈行番号〉|| |      |〈行番号〉|
            |                 |
            |GOTO〈行番号〉   |
```

解説

• 論理式の条件によってプログラム実行を制御します.

• 〈論理式〉が真(0以外)ならばTHENまたはGOTO以降を実行し，偽(0)ならばELSE以降が実行されます．もしELSE以降が省略されている場合，次の行が実行されます.

• IF...THEN...ELSE文において，THENやELSEに続けて別のIFを置いて多重にすることができます.

例 IF A$="y" THEN 200 ELSE END

• 文字変数A$が"y"のとき200行へ制御が移り，それ以外のときはプログラムの実行を終了します.

プログラム例

```
list↵
100 ' IF...THEN...ELSE/IF...GOTO...ELSE sample
110 INPUT A
120 FL=0
130 IF A<0 THEN A=ABS(A):FL=1
140 PRINT SQR(A);
150 IF FL THEN PRINT "i" ELSE PRINT
160 END
Ok
run↵
? 5↵
 2.23607
Ok
run↵
? -10↵
 3.16228 i
Ok
run↵
? -25↵
 5 i
Ok
```

• 単精度実数を読み込んで，その平方根を出力します.

• 負の数の場合130行で，その絶対値をとり，変数FLを1にしておきます.

• 150行でFLが1，つまり負の数が入力されたときは，答が虚数なので，"i"を出力しています.

INIT%

イニット・パーセント：initialize %

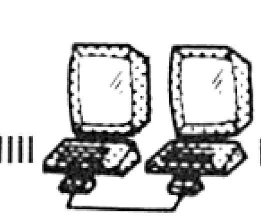
機能 RS-232C通信ポートを初期設定する

書式 INIT%〈ポート番号〉,〈モード〉,〈コマンド〉

解説
• RS-232C通信ポートおよびその入力バッファのイニシャライズを行います.
•〈ポート番号〉は1を指定します.
• RS-232C通信ポートの制御はμPD8251AFCというUSART(Universal Synchronous/Asynchronous Receiver/Transmitter)を使用しており,〈モード〉および〈コマンド〉でその通信方式を設定します. 詳細については, 資料6 μPD8251の使い方を参照してください.

例 INIT %1 , &HFB , &H37
•〈モード〉= FBH, 〈コマンド〉= 37Hで, μPD8251の設定を行い, 入力バッファを初期化します.

プログラム例


RS-232C

```
list
100 ' INIT% sample
110 INIT %1,&HFA,&H37
120 IF PORT(1) THEN PRINT INPUT$(PORT(1),%1);
130 C$=INKEY$:IF C$<>"" THEN PRINT %1,C$;
140 GOTO 120
Ok
```

• RS-232C通信ポートを通して入ってきたデータを画面に出力し, キー入力された文字をRS-232C通信ポートに出力します.
• 2台のPC-8001MKⅡSRをRS-232Cケーブル(リバース)でつないで, 両方でこのプログラムを実行させれば, コンピュータ同士で通信が行えます.
• 110行で, ストップビット2, 偶数パリティ, パリティイネーブル, キャラクタ長　7ビット, ボーレートファクタ　16, センド要求　ON, エラーリセット, 受信イネーブル, データターミナルレディ　ON, 送信イネーブルに初期設定しています.

参照：PRINT%, INPUT%, LINE INPUT%, INPUT$, PORT

資-105

INKEY$

インキー・ダラー：input key $

機能 押されたキーの文字を返す

書式 INKEY$

解説 • キー入力があればその文字を返します．キー入力がない場合はヌルストリングが返されます．

• キー入力された文字は画面に表示されません．

• INKEY$関数では STOP キーの入力を知ることはできません．INKEY$関数を含むステートメント実行中に STOP キーを押すと，プログラムの実行がとまるためです．

例 INKEY$ 返される値 （押されたキーの文字）

プログラム例

```
list
100 ' INKEY$ sample
110 PRINT "Enter any key to end.."
120 A$=INKEY$
130 IF A$="" THEN 110 ELSE END
Ok
run
Enter any key to end..
Enter any key to end..
Enter any key to end..
Enter any key to end..
Enter any key to end..
Enter any key to end..
Enter any key to end..
Enter any key to end..
Ok
```

• 何かキーを押すまで，"Enter any key to end.."を出力します．

• 実行例では8回"Enter any key to end.."を出力したところで，あるキーを押しています．

INP

アイ・エヌ・ピー：input

機能　入力ポートから得た値を返す

書式　INP(〈I/Oアドレス〉)

解説　•〈I/Oアドレス〉で指定された入力ポートから8ビットのデータを読み込み，それを関数の値として返します．

•〈I/Oアドレス〉として指定できる値の範囲は0～255です．

例　INP(&H33) 返される値 (33Hポートからの入力データ)

プログラム例

```
list
100 ' INP sample
110 PRINT "Type 'e' key to end.."
120 IF INP(2)=223 THEN END
130 DM$=INPUT$(1):PRINT DM$
140 GOTO 120
Ok
run
Type 'e' key to end..
j
i
o
l
i
j
e
Ok
```

•"e"のキーを押すまで，押したキーの文字を出力し続けます．

•120行は入力ポートの2番の値が，2進数で11011111になっているかどうかを判断しています．これは"e"のキーが押されたかどうかの判断です．

•実行例では，"j", "i", "o", "l", "i", "j", "e"の順に入力しています．

参照：OUT

INPUT

インプット：input

機　能　キーボードから入力したデータを指定した変数に代入する

書　式　INPUT〔"〈プロンプト文〉";〕〈変数名〉〔,〈変数名〉..〕

解　説

・INPUT文が実行されると，画面に疑問符(？)と1文字のスペースを表示し，キーボードからの入力待ちの状態になります．このとき，〈プロンプト文〉がある場合，〈プロンプト文〉に続いて疑問符を表示します．

・データは⏎キーを押すことによって入力され，変数に代入されます．〈変数名〉をコンマで区切って複数個指定した場合には，入力するデータもコンマで区切って〈変数名〉の数だけ入力するか，1つずつ⏎キーを押して入力します．この場合，2回目からは"？？"が表示され入力待ちとなります．

・〈変数名〉と入力されたデータの型が一致しなかったときには，"?Redo from start"と表示し，再び入力待ちになります．また，データが〈変数〉の個数より多い場合には"?Extra ignored"と表示し，余分なデータは捨てられます．

・⏎キーだけを押した場合，変数は現在の値が採用されます．つまり，INPUT文を実行する前の値となります．

・文字変数に文字列を代入する場合，コンマや文字列の前後の意味のある空白を入力したいときは，ダブルクォーテーション(")で文字列を囲みます．

・値としてダブルクォーテーション(")を入力することはできません．

例　INPUT"DATA＝";D$

・"DATA=? "と表示し，文字列の入力を待ちます．文字列が入力されると文字変数D$に代入します．

プログラム例

```
list⏎
100 ' INPUT sample
110 INPUT "ﾅﾏｴ ";NA$
120 INPUT "ｾｲﾍﾞﾂ ";SE$
130 INPUT "ﾈﾝﾚｲ ";OD
140 PRINT "ﾅﾏｴ    :",NA$
150 PRINT "ｾｲﾍﾞﾂ  :",SE$
160 PRINT "ﾈﾝﾚｲ   :",OD
170 END
Ok
run⏎
ﾅﾏｴ ? ﾅｶﾊﾗ ﾕｶﾘ⏎
ｾｲﾍﾞﾂ ? ｵﾝﾅ⏎
ﾈﾝﾚｲ ? ﾊﾀﾁ⏎
?Redo from start
ﾈﾝﾚｲ ? 20⏎
ﾅﾏｴ    :         ﾅｶﾊﾗ ﾕｶﾘ
```

```
セイベツ  :          オンナ
ネンレイ  :           20
Ok
```

・名前，性別，年齢を読み込んで，それを出力します．

・実行例では年齢を入力するときに数値を入力しなかったために，"?Redo from start"というメッセージが表示され，再び入力待ちになっています．

INPUT#

ディスク 〈ファイル番号〉が1～15の場合

インプット・シャープ：input #

機能 シーケンシャルファイルからデータを読み込む

書式 INPUT #〈ファイル番号〉,〈変数〉〔,〈変数〉...〕

解説 • フロッピィディスクまたはカセットテープに書き込まれたシーケンシャルファイルからデータを読み込みます．

• フロッピィディスクの場合，〈ファイル番号〉は，1～15の値をとります．これは，OPEN文によってオープンしたときに指定したファイル番号です．

• カセットテープでは，〈ファイル番号〉に－1を指定します．この場合，ファイルをオープンする必要はありません．

例 INPUT #－1,DT

• カセットテープから，数値データを読み込み変数DTに代入します．

プログラム例

ディスク

```
list
100 ' INPUT# sample
110 OPEN "2:animal" FOR INPUT AS #1
120 IF EOF(1) THEN END
130 INPUT #1,A$
140 PRINT A$
150 GOTO 120
160 END
Ok
run
ﾜﾝﾜﾝ
ﾆｬｰｵ
ﾋﾋｰﾝ
ﾋﾟｰﾋﾟｰ
Ok
```

• PRINT#文のプログラム例を実行することによって作成されたシーケンシャルファイルの内容を出力します．

• 120行ではファイルが終わりに達していたら実行を停止するようになっています．

注意：• フロッピィディスクを使用してデータを読み込む場合，書き込んだときのデータの型と一致するように〈変数〉を指定してください．また，カセットテープを使用してデータを読み込む場合は，書き込んだときのデータの型と数が一致するよう〈変数〉を指定します．（詳しくは，3章　データファイルを作るを参照してください）．

参照：OPEN，PRINT#，PRINT# USING

INPUT$

ディスク　〈ファイル番号〉が1～15の場合

インプット・ダラー：input $

機能　キーボード，ファイルなどから指定された長さの文字列を読み込みその値を返す

書式　INPUT$ (〈文字数〉〔，｜〔#〕〈ファイル番号〉｜〕)
｜%〈ポート番号〉｜

解説　・〈文字数〉は文字列の長さを指定するもので，0～255の値をとります．

・〈ファイル番号〉が省略された場合には，キーボードから指定された〈文字数〉だけ読み込みます．このとき，入力された文字は画面に表示されません．

・〈ファイル番号〉が指定されると，ファイルから〈文字数〉分の文字列を読み出します．〈ファイル番号〉は1～15の値を指定します．これはOPEN文でオープンされたときに指定したファイル番号です．

・〈ポート番号〉を指定するとRS-232Cの入力バッファより〈文字数〉分の文字列を読み出します．ただしINPUT$関数を実行する前には，INIT%文でRS-232C通信ポートを初期化しておきます．

・INPUT$関数は，指定された〈文字数〉の文字が入力されるのを待ち続けますが，既にバッファに入力済みのデータがある場合には，バッファの先頭から文字を拾ってきます．また，INPUT$関数は，CTRL ＋ C，STOP キーを除くすべての文字をそのまま読み出しますので，INPUT文やLINE INPUT文では入力することのできない制御文字等も入力することができます．

例　INPUT$ (5)　返される値⇒　(キーボードから5文字読み込んだ文字列)

プログラム例

```
list↵
100 ' INPUT$ sample
110 A$=INPUT$(1)
120 IF A$=CHR$(&H1B) THEN END
130 IF A$=CHR$(13) THEN PRINT ELSE PRINT A$;
140 GOTO 110
Ok
run↵
personal
Ok
```

・ESC キーが押されるまで，キーボードから入力した文字を表示します．

・110行で，キーボードから1文字読み込みます．

・120行は，入力した文字が ESC キーかどうか判断しています．

・↵キーを入力した場合，現在文字が表示されている行の先頭にカーソルが移動してしまうので，130行で改行するようにします．

• 実行例では"p", "e", "r", "s", "o", "n", "a","l"と入したあとに， ESCキーを入力しています.

参照：INPUT, LINE INPUT

INPUT%

インプット・パーセント：input %

機能 RS-232C通信ポートの入力バッファからデータを入力する

書式 INPUT%〈ポート番号〉,〈変数名〉〔,〈変数名〉...〕

解説

• RS-232C通信ポートの入力バッファよりデータを入力し，その値を〈変数名〉で指定された変数に代入します．〈ポート番号〉は1を指定します．

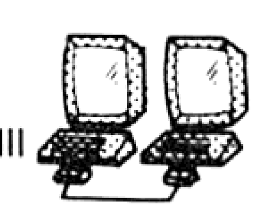
• 入力されるデータはCR(キャリッジリターン)コード(0DH)またはコンマ(，)で区切られます．バッファの先頭からこの区切りを探し，区切りが見つかったところで，それらを変数に代入します．もし，バッファ内にCRコードまたはコンマ(，)がなければ，RS-232Cチャネルを通してこの区切りが入力されるまで待ちます．

• INPUT%文を実行する前にRS-232C通信ポートは，INIT%文で初期化しておきます．

例 INPUT %1, A$

• RS-232C通信ポートの入力バッファよりデータを入力し，変数A$に代入します．

プログラム例



RS-232C

```
list
100 ' INPUT% sample
110 INIT %1,&HFA,&H37
120 A$="":INPUT %1,A$
130 IF A$="end" THEN OUT &HE6,0:END
140 PRINT A$
150 GOTO 120
Ok
```

• RS-232C通信ポートからデータを受け取って，画面に出力します．

• 120行で，RS-232C通信ポートからのデータを受け取り，変数A$に代入しています．

• 受け取ったデータが"end"であれば，130行でプログラムの実行を終了し，"end"でなければ140行でそのデータを画面に出力します．

• これは受信側のプログラムです．送信側のプログラム例はPRINT%文を参照してください．

参照：INIT%，PRINT%，PORT

INSTR

インストリング：instring

機能 文字列の中から任意の文字列を探し，その文字の位置を返す

書式 INSTR(〔〈数式〉,〕〈文字列1〉,〈文字列2〉)

解説 •〈文字列1〉の中から〈文字列2〉を探し，見つかればその位置を返します．このとき，〈数式〉は探し始める位置を指定し，省略した場合は〈文字列1〉の先頭から探し始めます．

•〈文字列2〉が見つからなかったとき，〈文字列1〉がヌルストリングであったとき，〈数式〉が〈文字列1〉の長さより大きいときは0を値として返します．

•〈文字列2〉にヌルストリングを指定すると，〈数式〉と同じ値を返します．

例 INSTR (TX$, " the ") 返される値⇒ (TX$に含まれる"the"の位置)

プログラム例

```
list↵
100 ' INSTR sample
110 A$="abcdefghijklmnopqrstuvwxyz"
120 ALP$=INPUT$(1)
130 ALP=ASC(ALP$)
140 IF ALP>64 AND ALP<91 THEN ALP=ALP+32
150 A=INSTR(A$,CHR$(ALP))
160 IF A=0 THEN PRINT ALP$;" ﾊ ｱﾙﾌｧﾍﾞｯﾄ ﾃﾞﾊｱﾘﾏｾﾝ":END
170 PRINT USING "! ﾊ ## ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ";ALP$,A
180 END
Ok
run↵
t ﾊ 20 ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ
Ok
run↵
Z ﾊ 26 ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ
Ok
```

•キーボードから入力した文字が，アルファベットの何番目にあるかを出力します．

•150行で，入力されたアルファベットが何番目にあるか調べています．この例では小文字で調べているので，入力されたアルファベットが大文字の場合は140行で，大文字から小文字に変換しています．

•実行例では"t"と"Z"を入力しています．

INT

インテジャー：integer

機　能　小数点以下を切り捨てた整数値を返す

書　式　INT (〈数式〉)

解　説　• 〈数式〉の値を超えない最大の整数値(小数点以下を切り捨てた値)を返します.

例

INT (1.23) 返される値 ⇒ 1

INT (−3.21) 返される値 ⇒ −4

プログラム例

```
list
100 ' INT sample
110 FOR A=-2 TO 2 STEP .5
120 PRINT "A =";A,"FIX(A) =";FIX(A),"INT(A) =";INT(A)
130 NEXT A
140 END
Ok
run
A =-2           FIX(A) =-2      INT(A) =-2
A =-1.5         FIX(A) =-1      INT(A) =-2
A =-1           FIX(A) =-1      INT(A) =-1
A =-.5          FIX(A) = 0      INT(A) =-1
A = 0           FIX(A) = 0      INT(A) = 0
A = .5          FIX(A) = 0      INT(A) = 0
A = 1           FIX(A) = 1      INT(A) = 1
A = 1.5         FIX(A) = 1      INT(A) = 1
A = 2           FIX(A) = 2      INT(A) = 2
Ok
```

• いくつかの実数に対して，FIX関数とINT関数を実行した結果を出力します.

• このプログラム例はFIX関数と同じものです.

注意：• この関数はCINT関数と異なり，数値の型変換は行いません.

参照：FIX, CINT

KEY

キー：key

機能 ファンクションキーの内容を定義する

書式 KEY〈キー番号〉,〈文字列〉

解説

- ファンクションキーの内容を定義します.
- 各ファンクションキーには，最大15文字までの文字列(コントロール文字を含む)を定義することができます.
- キーボードから直接入力できない文字は，CHR$関数をつないで使います.
- 〈キー番号〉は1～10で指定します.
- システム起動時，ファンクションキーには

f・1・・・load "	f・6・・・save "
f・2・・・width	f・7・・・screen
f・3・・・go to	f・8・・・print
f・4・・・list	f・9・・・list. C_R ↑↑
f・5・・・run C_R	f・10・・・cont C_R

が定義されています.

例 KEY 1,"width 80"+CHR$ (13)

- ファンクションキー1に，"width 80C_R"という内容を定義します.

プログラム例

```
list
100 ' KEY sample
110 KEYLIST
120 FOR I=1 TO 5
130   READ K$:KEY I,K$
140 NEXT I
150 KEYLIST
160 END
170 DATA "auto ","files ","gosub ","list ","run"
Ok
run

load "          save "
width           screen
go to           print
list            list.CR↑↑
runCR           contCR

auto            save "
files           screen
gosub           print
list            list.CR↑↑
run             contCR
Ok
```

• 現在のファンクションキーの内容と，定義し直した後のファンクションキーの内容を表示します.

• 130行のKEY文で，I番目のファンクションキーに，170行のデータのI番目の文字列を割り当てます.

参照：KEY LIST, CHR$

KEY LIST

キー・リスト：key list

機能 ファンクションキーの内容を表示する

書式 KEY LIST

解説
- 現在，ファンクションキーに定義されている内容を画面上に表示します．
- コントロールコードも表示します．

例 KEY LIST
- ファンクションキーの内容を表示します．

実行例

```
key list

load "              save "
width               screen
go to               print
list                list. CR↑↑
run CR              cont CR
Ok
```

- ファンクションキーの内容を表示します．

参照：KEY

KILL

ディスク

キル：kill

機能 フロッピィディスク上のファイルを削除する

書式 KILL〈ファイル名〉

解説
• 〈ファイル名〉で指定したファイルをフロッピィディスクから削除します.

• 削除するファイルはクローズされていなければなりません. もし, オープン状態のファイルを削除しようとすると "File already open" エラーとなります. また, 書き込み禁止属性の付いたファイルを削除しようとすると "File write protected" エラーとなります. この場合SET文で書き込み禁止属性を解除してからKILLコマンドを実行してください.

• KILLコマンドは, すべてのディスクファイル(プログラムファイル, ランダムファイル, シーケンシャルファイル)に対して使用できます.

• ドライブ番号はファイルの前に "n:" と指定します. 省略されたときドライブ1が指定されたとみなされます.

例 KILL "2 : demo"

• ドライブ2に入っているフロッピィディスクの "demo" というファイルを削除します.

プログラム例

ディスク

```
list
100 ' KILL sample
110 OPEN "2:data3" FOR OUTPUT AS #1
120 PRINT #1,"KILL statement test"
130 CLOSE
140 FILES 2:PRINT
150 KILL "2:data3"
160 FILES 2
170 END
Ok
run
data1       1    data2       1    data4       1    data5       1    data6       1
data7       1    data8       1    data9       1    addres s    1    rdata       1
animal      1    test   .nsr 1    test2 .nsr  1    test3  nsr  1    test4 .nsr  1
newtes t    1    fdata       1    usr    .    1    data3       1

data1       1    data2       1    data4       1    data5       1    data6       1
data7       1    data8       1    data9       1    addres s    1    rdata       1
animal      1    test   .nsr 1    test2 .nsr  1    test3  nsr  1    test4 .nsr  1
newtes t    1    fdata       1    usr    .    1
Ok
```

• ドライブ2に入っているフロッピィディスクに "data3" というファイルを作り, その後そのファイルを削除します.

• 110行〜130行で作成したファイルを150行で削除しています.

• ファイルを作成した後と, 削除した後にFILESコマンドを行っています.

参照：SET

LEFT$

レフト・ダラー：left $

機能 文字列の左側から指定された長さの文字列を返す

書式 LEFT$(〈文字列〉,〈数式〉)

解説
- LEFT$関数は〈文字列〉の左側から〈数式〉で指定された長さの文字列を取り出し，それを値として返します．
- 〈数式〉の範囲は0～255です．
- 〈数式〉が〈文字列〉の総文字数以上の場合は〈文字列〉全体を，また〈数式〉が0ならばヌルストリングを返します．

例 LEFT$(TIME$,2) 返される値 (TIME$変数の左側2文字)
- カレンダ時計の"時"だけを取り出します．

プログラム例

```
list
100 ' LEFT$ sample
110 INPUT A$
120 FOR I=1 TO 10
130   PRINT LEFT$(A$,I)
140 NEXT I
150 END
Ok
run
? 0123456789
0
01
012
0123
01234
012345
0123456
01234567
012345678
0123456789
Ok
```

- 読み込んだ文字列の左から1番目まで，2番目まで……10番目までの文字を出力します．
- 130行で，A$の左からI番目までの文字を出力します．

参照：RIGHT$, MID$

LEN

レングス：length

機　能　文字列の長さを返す

書　式　LEN(〈文字列〉)

解　説

• 与えられた文字列の長さの値を返します.

• LEN関数は普通の文字の他に，空白や，文字として表示されない文字も数えます.

例　LEN(TIME$) 返される値 ⇒ 8（＝TIME$変数の長さ）

プログラム例

```
list↵
100 ' LEN sample
110 INPUT A$:L=LEN(A$)
120 GOSUB 160
130 PRINT "* ";A$;" *"
140 GOSUB 160
150 END
160 ' frame
170 FOR I=1 TO L+4
180   PRINT "*";
190 NEXT I
200 PRINT
210 RETURN
Ok
run↵
? NEC Personal Computer↵
*************************
* NEC Personal Computer *
*************************
Ok
```

• 読み込んだ文字列を"＊"で囲んで出力するプログラムです.

• 110行で，入力した文字列の長さを変数Lに代入しています.

• 160行から210行までは，入力した文字列の長さより"＊"を4つ多くを出力するサブルーチンです．120行と140行でこのサブルーチンを呼び出して枠を作っています.

LET

レット：let

機能 変数に値を代入する

書式 〔LET〕〈変数名〉＝〈式〉

解説
- LETは省略できます.
- 〈式〉の内容は数値，文字列のいかなる型であってもかまいませんが,〈変数名〉と〈式〉の値の型が一致しなければなりません．つまり，右辺が数値式のとき左辺は数値変数，右辺が文字式のとき左辺は文字変数でなければなりません.

例 PI＝3.14159
- 数値変数PIに，3.14159という数値を代入します.

プログラム例

```
list
100 ' LET sample
110 LET A=123
120 LET B=.21
130 C=A+B
140 PRINT "A  =";A
150 PRINT "B  =";B
160 PRINT "A+B=";C
170 END
Ok
run
A  = 123
B  = .21
A+B= 123.21
Ok
```

- 変数A, Bにそれぞれ123，0.21を代入し，その和を求めて出力します.
- 130行では，LETを省略した代入文を使っています.

LINE

ライン：line

機能 グラフィック画面に直線や長方形を描く

書式

```
LINE〔 |(gx1,gy1)      |〕— |(gx2,gy2)      |〔,〈パレット番号〉〕
       | STEP(sx1,sy1) |     | STEP(sx2,sy2) |
    〔,| B |〕
       |BF |
```

解説 •グラフィック座標系の2点(gx1, gy1), (gx2, gy2)を結ぶ直線を描きます．指定できる範囲は−16384〜16383までです．(gx1, gy1)が省略された場合，LPの値をとります．

•線の色は，〈パレット番号〉によって指定します．省略された場合は，COLOR文で指定された〈フォアグラウンドカラー〉が用いられます．

•"B"および"BF"は，指定した2点を対角とする長方形を描きます．"B"はBoxの意味で，枠のみを描きます．また，"BF"はBox Fillの意味で，その長方形を塗りつぶします．

例 LINE(100,100)—(200,150),1,B

•グラフィック座標(100，100)，(200，150)の2点を対角とする長方形をパレット番号1の色で描きます．

プログラム例

```
list⏎
100 ' LINE sample
110 SCREEN 0,0:CLS 3
120 X1=RND(1)*639:Y1=RND(1)*199
130 X2=RND(1)*639:Y2=RND(1)*199
140 CL=RND(1)*6+1:MD=INT(RND(1)*3+1)
150 ON MD GOTO 160,170,180
160 LINE(X1,Y1)-(X2,Y2),CL:GOTO 190
170 LINE(X1,Y1)-(X2,Y2),CL,B:GOTO 190
180 LINE(X1,Y1)-(X2,Y2),CL,BF
190 IF INKEY$="" THEN 120
200 END
Ok
```

•何かキーを押すまで，いろいろな線や四角形を描きます．

•120行から140行で，乱数を使って座標や色を求めています． MDという変数には，1から3までの数が入り，その値によって直線，四角形，塗りつぶした四角形を描きます．

参照：COLOR

LINE@

ライン・アットマーク：line @

機能 テキスト画面を行単位に機能設定する

書式 LINE@〈行指定〉,〈ファンクションコード〉

解説 •テキスト画面のカラーモードにおいて，行単位に機能を設定します．

•〈行指定〉は0から(行数)－1までの値をとり，〈ファンクションコード〉は次のようになっています．

0または4 －ノーマル
1または5 －ブリンク
2または6 －リバース
3または7 －リバースブリンク

例 LINE@ 7,1

•テキスト画面の8行目をブリンクさせます．なお，この命令を白黒モードで実行すると "Syntax error" となります．

プログラム例

```
list
100 ' LINE@ sample
110 CONSOLE 0,25,1,1:WIDTH 80,20
120 FOR I=0 TO 7
130   COLOR I
140   FOR J=0 TO 7
150     PRINT "LINE@TEST";
160   NEXT J
170   PRINT
180   LINE@ I,I
190 NEXT I
200 END
Ok
```

•テキスト画面の0行目から7行目までに，機能設定を行います．

•110行で，テキスト画面をカラーモードにし，表示桁数を80桁にします．

•140行から160行で，130行で指定された色で "LINE@TEST" という文字列を8回続けて出力します．

•180行で，上で出力した行に機能Iを設定します．

LINE INPUT

ライン・インプット：line input

機能 1行全体(255文字以内)を区切ることなく文字変数に入力する

書式 LINE INPUT〔"〈プロンプト文〉";〕〈文字変数〉

解説
- 〈プロンプト文〉は，入力データを受ける前に画面に表示される文字列です．
- [↵] キーの入力までにキーボードから入力された文字列データが文字変数に代入されます．なお，文字列データを入力せず[↵]キーだけを押した場合，文字変数はヌルストリングが入力されたとみなされます．
- LINE INPUT 文の実行は，[CTRL]＋[C]，または[STOP]キーによって中断することができます．その場合，コマンドレベルに戻りますが，このときCONTコマンドを入力するとLINE INPUT 文の始めから実行が再開されます．

例 LINE INPUT " Name ? " ; NA$

- "Name?" を表示し，文字列の入力を待ちます．文字列が入力されると，それをNA$に代入します．

プログラム例

```
list↵
100 ' LINE INPUT sample
110 INPUT "INPUT : ";A$
120 LINE INPUT "LINE INPUT : ?";B$
130 PRINT:PRINT "INPUT : ",A$
140 PRINT "LINE INPUT : ",B$
150 END
Ok
run↵
INPUT : ? abcd,ABCD,1234↵
?Extra ignored
LINE INPUT : ?abcd,ABCD,1234↵

INPUT :       abcd
LINE INPUT :  abcd,ABCD,1234
Ok
```

- INPUT 文とLINE INPUT 文で文字列を入力し，画面に出力します．
- 110行でINPUT文，120行でLINE INPUT 文を使って文字列を読み込みで，130行と140行で出力しています．
- 実行例では，INPUT 文で1個の変数を指定してあるにもかかわらず，3個のデータが入力されたため，"?Extra ignored" というメッセージを表示しています．この場合，最初のデータは変数A$に代入されますが，残りの2個のデータは無視されます．

LINE INPUT#

ディスク

ライン・インプット・シャープ：line input #

機　能　1行全体(255文字以内)を区切ることなくシーケンシャルファイルから文字変数に代入する

書　式　LINE INPUT#〈ファイル番号〉,〈文字変数〉

解　説

• 〈ファイル番号〉はOPEN文によって指定したファイル番号です．

• 〈文字変数〉は，行全体が代入される文字変数名です．

• LINE INPUT#文はシーケンシャルファイルから，CRコードまでの，すべての文字を区切ることなく読み込みます．読み込める文字数は255文字までです．

• LINE INPUT#文はデータファイルのそれぞれの行がいくつかの欄に分割されているときや，アスキーセーブされたプログラムをデータとして読み込むときなどに有効です．

例　LINE INPUT#1, T$

• ファイル番号1でオープンしたファイルから1行分のデータを読み込み，変数T$に代入します．

プログラム例

ディスク

```
list
100 ' LINE INPUT# sample
110 OPEN "2:data2" FOR INPUT AS #1
120 IF EOF(1) THEN END
130 LINE INPUT #1,A$
140 PRINT A$
150 GOTO 120
Ok
run
 In some cases,business computerists become
so incensed as to wage a mental war with s-
oftware,determined to succeed in  spite  of
the documentation. I know of none business-
man who actually refused to look at the do-
cumentation that accompanies  any  software
he acquires. When I asked him why,he smiled
slyly,"It's worthless." Now there  is a man
obsessed.
Ok
```

• ドライブ2に入っているフロッピィディスクの"data2"というファイルの内容を表示します．

• 130行のLINE INPUT#文では，変数A$にCRコードの前までのすべての文字列が入ります．

参照：OPEN

LINE INPUT%

ライン・インプット・パーセント：line input %

機能　1行全体(255文字以内)を区切ることなくRS-232C通信ポートの入力バッファから文字列データを入力する

書式　LINE INPUT%〈ポート番号〉,〈文字変数〉

解説　•〈ポート番号〉で指定されたRS-232C通信ポートの入力バッファから文字列データを入力し，そのデータを〈文字変数〉で指定される変数に代入します．〈ポート番号〉は1を指定します．

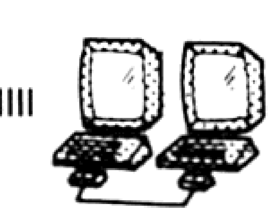
•入力されるデータの区切りはCRコードです．

•LINE INPUT%文を実行する前に，RS-232C通信ポートはINIT%文で初期化しておきます．

例　LINE INPUT %1,T$

•RS-232C通信ポートの入力バッファから，1行分のデータを読み込み，変数T$に代入します．

プログラム例



RS-232C

```
list
100 ' LINE INPUT% sample
110 INIT %1,&HFA,&H37
120 LINE INPUT %1,A$
130 PRINT A$
140 GOTO 120
Ok
```

•RS-232C通信ポートからデータを受け取って画面に出力します．INPUT%文のプログラム例との違いは，CRコードのみを区切り記号として扱っていることです．

•INPUT%文のプログラム例も参照してください．

参照：INIT%, PRINT%, INPUT%

LIST/LLIST

リスト/エル・リスト：list/lpt list

機能 プログラムの内容を出力する

書式 (1) LIST〔〈行番号〉〕〔－〔〈行番号〉〕〕

(2) LLIST〔〈行番号〉〕〔－〔〈行番号〉〕〕

解説 • プログラムの内容を画面に表示したり，プリンタにプリントアウトします．LIST コマンドは画面に，LLIST コマンドはプリンタに出力します．

• パラメータを省略すると，プログラムの全部が出力されます．

• 〈行番号〉を１つだけ指定するとその行のみが出力されます．

• ２つの〈行番号〉をハイフン(－)でつないで指定すると，指定した範囲のプログラムが出力されます．このとき，最初の〈行番号〉を省略した場合は，プログラムの先頭からハイフンに続く〈行番号〉までが出力されます．後の〈行番号〉を省略した場合は，指定された〈行番号〉からプログラムの終わりまでが出力されます．

• LIST/LLIST コマンドは実行し終わると，コマンドレベルに戻ります．

例 LIST －190

• プログラムの先頭から190行までを画面に表示します．

実 行 例

(a)
```
list↵
100 ' LIST sample
110 N=1
120 ' start
130 PRINT N
140 N=N+1
150 IF N<10 THEN 140
160 END
Ok
```
(b)
```
list 150↵
150 IF N<10 THEN 140
Ok
```
(c)
```
list 120-150↵
120 ' start
130 PRINT N
140 N=N+1
150 IF N<10 THEN 140
Ok
```
(d)
```
list -140↵
100 ' LIST sample
110 N=1
120 ' start
130 PRINT N
140 N=N+1
Ok
```
(e)
```
list 140-↵
140 N=N+1
150 IF N<10 THEN 140
160 END
Ok
```

(a) プログラムを全部出力します.

(b) 150行だけを出力します.

(c) 120行から150行までを出力します.

(d) プログラムの先頭から140行までを出力します.

(e) 140行からプログラムの終わりまでを出力します.

LOAD

ディスク

ロード：load

機能 フロッピィディスクからプログラムをロードする

書式 LOAD〈ファイル名〉〔,R〕

解説
• 〈ファイル名〉で指定されたプログラムファイルをメモリ上にロードします．このとき，以前メモリ上にあったプログラムは消され，変数の値も初期化されます．また，開いているファイルがあった場合は，すべて閉じられます．

• Rオプションを付けると，ファイルを開いたままプログラムをロードし，ただちに実行を開始します．

• LOADコマンドは，指定されたファイルを見つけて実際にプログラムのロードを開始するまでは，メモリ中のプログラムを保存します．

例 LOAD "2:demo"

• ドライブ2に入っているフロッピィディスクから"demo"というファイル名の付いたプログラムをロードします．

実行例

ディスク

(a)
```
load "2:test.nsr"
Ok
```
(b)
```
load "2:test.nsr",r
```

• (a)はドライブ2に入っているフロッピィディスクから"test. nsr"というファイルをロードします．

• (b)は，現在開いているファイルを開いたままで，ドライブ2に入っているフロッピィディスクから"test. nsr"というファイルをロードして実行を開始します．

参照：SAVE, MERGE, RUN

LOC

ディスク

エル・オー・シー：location counter of a file

機能 フロッピィディスクファイル中での論理的な現在位置を返す

書式 LOC(〈ファイル番号〉)

解説
- 〈ファイル番号〉で指定されたファイルがランダムファイルであった場合には，そのランダムファイルに対して最後に読み書きされたレコード番号(GET#文，PUT#文で指定した番号)を返します．
- 〈ファイル番号〉で指定されたファイルが，シーケンシャルファイルであった場合には，そのファイルがオープンされてから，読み書きされたレコード数を返します．ここで言うレコード数とは，セクタ数をさします．

例 LOC(2) 返される値 ⇨ (ファイル番号2でオープンされたファイルの現在位置)

プログラム例

ディスク

```
list ↵
100 ' LOC sample
110 OPEN "2:data6" AS #1
120 FIELD #1,20 AS A$,8 AS B$
130 INPUT "ﾅﾏｴ";NA$:INPUT "ｷｭｳﾖ";KY#
140 LSET A$=NA$:RSET B$=MKD$(KY#):PUT #1
150 PRINT NA$;" ｻﾝ ﾉ ﾃﾞｰﾀ ﾊ";LOC(1);"ﾊﾞﾝ ﾃﾞｽ｡"
160 INPUT "continue?(y/n)";C$
170 IF C$="y" THEN PRINT:GOTO 130
180 IF C$<>"n" THEN 160 ELSE END
Ok
run ↵
ﾅﾏｴ? ﾃﾗﾀﾞ↵
ｷｭｳﾖ? 150000↵
ﾃﾗﾀﾞ ｻﾝ ﾉ ﾃﾞｰﾀ ﾊ 1 ﾊﾞﾝ ﾃﾞｽ｡
continue?(y/n)? n↵
Ok
```

- ランダムファイルを作成しながら，登録したレコード番号を出力します．
- 140行で書き込まれたレコード番号を，150行で出力しています．

参照：OPEN, PRINT#, PUT#

LOCATE

ロケート：locate

機能 カーソル位置を移動させる

書式 LOCATE〈cx〉,〈cy〉〔,〈カーソルスイッチ〉〕

解説
- カーソル位置をテキスト画面のキャラクタ座標〈cx〉,〈cy〉へ移動します.
- 〈cx〉は水平座標を表し,〈cy〉は垂直座標を表します.
- 〈cx〉,〈cy〉は,テキスト画面の左上を(0,0)とするキャラクタ座標により指定します.これらの値をとる範囲は,WIDTH文で指定された〈桁数〉と〈行数〉によって決まります.

〈cx〉＝ 0～(〈桁数〉－1) 白黒モード
　　　 0～(〈桁数〉－2) カラーモード

〈cy〉＝0～(〈行数〉－1)

- 〈カーソルスイッチ〉は,カーソルの状態を決めるスイッチで,1で表示され,0で表示されなくなります.

例

LOCATE 0,10

- キャラクタ座標(0,10)にカーソルを移動させます.

LOCATE 0,0,0

- 画面左上にカーソルを移動させ,さらにカーソルを消します.

プログラム例

```
list
100 ' LOCATE sample
110 CONSOLE 0,25,0,1:WIDTH 80,25:CLS
120 INPUT "x,y";X,Y
130 LOCATE X,Y:PRINT "LOCATE"
140 GOTO 120
Ok
```

- テキスト画面の座標(x, y)を読み込んで,その位置に "LOCATE" を出力します.
- 130行で,xおよびyで指定された位置にカーソルを移動させて "LOCATE" を出力しています.

LOF

ディスク

エル・オー・エフ：length of a file

機能 ファイルの大きさを返す

書式 LOF(〈ファイル番号〉)

解説 •〈ファイル番号〉により指定されたディスクファイルの大きさをセクタ数で返します．そのファイルがランダムファイルであれば，ファイルに書き込むときPUT#文で指定した最大のレコード番号に相当します．

•GET#文でLOF関数の返す値よりも大きなレコード番号を指定すると，最大のレコード番号より大きな値を指定したことになり，"Input past end"エラーとなります．

例 LOF(2) 返される値 ⇒ (ファイル番号2でオープンしたファイルの大きさ)

プログラム例

ディスク

```
list
100 ' LOF sample
110 OPEN "2:data7" FOR OUTPUT AS #1
120 PRINT #1,123:CLOSE
130 OPEN "2:data7" FOR INPUT AS #1
140 OPEN "2:data8" AS #2
150 FIELD #2,128 AS A$,128 AS B$
160 PRINT "data7 ﾉ ｾｸﾀ  ｽｳ ﾊ";LOF(1);"ﾃﾞｽ｡"
170 PRINT "data8 ﾉ ﾚｺｰﾄﾞｽｳ ﾊ";LOF(2);"ﾃﾞｽ｡"
180 END
Ok
run
data7 ﾉ ｾｸﾀ  ｽｳ ﾊ 1 ﾃﾞｽ｡
data8 ﾉ ﾚｺｰﾄﾞｽｳ ﾊ 0 ﾃﾞｽ｡
Ok
```

•ドライブ2に入っているフロッピィディスクに"data7"と"data8"というファイルを作り，その大きさを表示します．

•"data7"では120行で123というデータを書き込んでいるため，セクタ数が1になりますが，"data8"ではデータを書き込んでいないためレコード数は0になります．

LOG

ログ：logarithm

機能 自然対数を返す

書式 LOG(〈数式〉)

解説

- 〈数式〉によって与えられた値の自然対数(e=2.71828を底とした対数)を返します.
- 〈数式〉には正の数を指定します.
- LOG関数の演算は単精度で行われます.

例

LOG(100) 返される値 4.60517

LOG(1) 返される値 0

LOG(—5) 返される値 " Illegal function call "エラー

プログラム例

```
list
100 ' LOG sample
110 INPUT X
120 A=LOG(10)
130 B=LOG(X)
140 PRINT "log10(";X;")=";B/A
150 END
Ok
run
? 100
log10( 100 )= 2
Ok
run
? .1245
log10( .1245 )=-.904831
Ok
```

- 自然対数を用いて常用対数を計算します.
- xが与えられたときの$\log_{10}X$は,

$$\log_{10}(X)=\log_e(X)/\log_e(10)$$

で与えられます.

LPOS

ライン・ポジション：line position

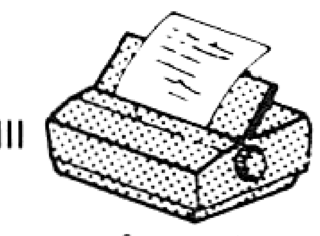
機能 現在のプリンタのヘッド位置を返す

書式 LPOS(〈式〉)

解説
- 現在，プリンタのヘッドがどの位置にあるかを桁数で返します.
- 〈式〉の値は意味を持たず，通常は0を指定します.

例 LPOS(0) 返される値→（プリンタのヘッド位置）

プログラム例


プリンタ

```
list
100 ' LPOS sample
110 FOR I=0 TO 20
120   IF LPOS(0)>=40 THEN LPRINT
130   LPRINT SQR(I),
140 NEXT I
150 END
Ok
run
Ok
```

```
0              1              1.41421
1.73205        2              2.23607
2.44949        2.64575        2.82843
3              3.16228        3.31663
3.4641         3.60555        3.74166
3.87298        4              4.12311
4.24264        4.3589         4.47214
```

- 0から20までの平方根を，プリンタに出力します.
- 120行でプリンタのヘッドが40桁目を超えた場合は，改行するようにします.

注意：• LPOS関数の値はBASICが管理している値で，プリンタによっては，実際のヘッド位置とこの位置とが異なる場合があります.

参照：POS

LSET/RSET

ディスク

エル・セット/アール・セット：left set/right set

機能 ファイルバッファへデータを書き込む

書式 LSET〈文字型変数〉＝〈文字列〉

RSET〈文字型変数〉＝〈文字列〉

解説 • FIELD文で定義した文字型変数に該当するファイルバッファ領域にデータを格納します．

• 格納するデータは文字型データです．数値データは，MKI$, MKS$, MKD$関数のいずれかにより文字型データに変換してから格納します．

• 〈文字列〉の長さが，FIELD文で割り当てられた長さより短い場合，LSET文では左詰め，RSET文では右詰めでフィールドを満たし，余った部分には空白(CHR$(&H20))が代入されます．また〈文字列〉がフィールドより長い場合は，LSET文，RSET文とも右側の部分が無視されます．

• MKI$, MKS$, MKD$関数によって返された〈文字列〉の長さが，FIELD文で定義された文字型変数の長さより短い場合は，LSET文を使います．

• 〈文字型変数〉は，FIELD文であらかじめ定義されていなければなりません．定義されていない変数を用いるとエラーとなります．

例 LSET A$＝"BASIC"

• FIELD文でファイルバッファに割り当てられた文字型変数A$の領域に，"BASIC"というデータを左詰めで書き込みます．

プログラム例

ディスク

```
list
100 ' LSET/RSET sample
110 OPEN "lrdata" AS #1
120 FIELD #1,10 AS A$,10 AS B$
130 INPUT C$
140 LSET A$=C$
150 RSET B$=C$
160 PUT #1,1
170 GET #1,1
180 PRINT A$:PRINT B$
190 END
Ok
run
? ABC
ABC
       ABC
Ok
```

• 文字列を読み込み，それを左詰めと右詰めで書き込みます．

• 140行で左詰め，150行で右詰めで読み込んだデータを格納しています．

参照：FIELD, PUT

MERGE

ディスク

マージ：merge

機能 メモリ上にあるプログラムにフロッピィディスク上のプログラムを結合する

書式 MERGE〈ファイル名〉

解説
• メモリ上のプログラムに〈ファイル名〉で指定したプログラムをメモリ上で結合します．
• 指定するプログラムファイルは，前もってアスキーセーブしたものでなければなりません．
• ファイル中のプログラムと，メモリ上のプログラムに同一行番号があった場合には，メモリ上の行はファイルの中の行に置き換わります．
• MERGE コマンドは実行を終了すると必ずコマンドレベルに戻ります．

例 MERGE "2:result.asc"

• ドライブ2に入っている"result. asc"というファイル名のプログラムファイルを，メモリ上にあるプログラムに結合します．

実行例

ディスク

```
merge "2:test3.nsr"↵
Ok
```

• 現在，メモリ上にあるプログラムとドライブ2に入っているフロッピィディスクにアスキーセーブされている"test3. nsr"というプログラムを結合します．

参照：LOAD, SAVE

MID$

ミッド・ダラー：middle $

機能 文字列の一部を指定された文字列で置き換える

書式 MID$(〈文字変数〉,〈式1〉〔,〈式2〉〕)＝〈文字列〉

解説
- 〈文字変数〉の〈式1〉番目の文字から〈式2〉個の文字を，〈文字列〉の最初から〈式2〉個の文字列で置き換えます．
- 〈式1〉の値は0より大きく，〈文字変数〉の長さ以下でなければなりません．
- 〈式2〉が省略された場合，〈式2〉の値を〈文字列〉の文字数より多く指定した場合，〈式2〉の値が〈式1〉番目から右の文字数(残りの文字数)より大きい場合，〈式2〉の値は(〈文字変数〉の長さ)−(〈式1〉−1)とみなされます．
- 〈文字変数〉にヌルストリングを指定することはできません．

例

A$＝" ABCDEF "

MID$(A$,4,3)＝"XYZ "

- 変数A$の4文字目から3文字("DEF")を"XYZ"に置き換えます．

プログラム例

```
list
100 ' MID$ sample
110 DATA My name is !!!!!!!!
120 DATA Kako,Aikawa,Terada,Nakahara
130 READ A$
140 FOR I=1 TO 4
150   READ NA$:MID$(A$,12)=NA$
160   PRINT A$
170 NEXT
180 END
Ok
run
My name is Kako!!!!
My name is Aikawa!!
My name is Terada!!
My name is Nakahara
Ok
```

- 120行に用意してある名前を読み込んで，"My name is……"という文を出力するプログラムです．
- 150行のMID$文で，変数A$の12番目の文字以降，つまり"!!!!!!!!"を，変数NA$に読み込んだ文字列で必要な文字数だけ置き換えています．

MID$

ミッド・ダラー：middle $

機能 文字列の指定された位置から任意の長さの文字列を返す

書式 MID$(〈文字列〉,〈式 1〉〔,〈式 2〉〕)

解説
- 〈文字列〉の〈式 1〉番目から〈式 2〉個の文字列を取り出し，値として返します．
- 〈式 1〉は 1 ～255，〈式 2〉は 0 ～255の範囲の値をとります．
- 〈式 2〉を省略したとき，または〈式 2〉が〈式 1〉番目から右の文字数(つまり，残りの文字数)より大きくなったとき，〈式 1〉番目より右の文字列全部が結果として返されます．
- 〈式 2〉が 0 のとき，または〈文字列〉全体の文字数が〈式 1〉より小さい場合，MID$関数はヌルストリングを返します．

例 MID$(TIME$,4,2) 返される値 ⇒ (分)

- TIME$変数の 4 文字目から 2 文字(＝分)を返します．

プログラム例

```
list
100 ' MID$ sample
110 WIDTH 40,25:CLS
120 A$="NEC Computer"+STRING$(10," ")
130 LOCATE 9,10:GOSUB 190
140 PRINT A$
150 B$=LEFT$(A$,1)
160 A$=MID$(A$,2)+B$
170 GOTO 130
180 ' wait routine
190 FOR I=1 TO 100 :NEXT
200 RETURN
Ok
```

- "NEC Computer" という文字を次々と左に流して表示します．
- このプログラムでは，"NEC Computer　　　　　　　　　　" という文字列を同じ場所で左に回転させながら表示し，流れていくように見せています．

MKI$/MKS$/MKD$

ディスク

エム・ケー・アイ・ダラー：make integer $
エム・ケー・エス・ダラー：make single $
エム・ケー・ディ・ダラー：make double $

機　能　数値を文字コードに変換して返す

書　式　MKI$(〈整数表記〉)
MKS$(〈単精度表記〉)
MKD$(〈倍精度表記〉)

解　説　• これらの関数は，数値をランダムファイルバッファに対してLSET/RSET文で書き込む際に使用します．

• 数値から文字列への変換は数値が持つ内部表現(２進数表現)の値をそのままそれに対応する文字コードにすることによって行われます．この逆の動作をする関数として，CVI/CVS/CVD関数が用意されています．

• 各関数の機能は次のとおりです．

MKI$－整数値を２文字(２バイト)の文字列に変換します．
MKS$－単精度数値を４文字(４バイト)の文字列に変換します．
MKD$－倍精度数値を８文字(８バイト)の文字列に変換します．

例　MKI$(16961) 返される値 "AB"

• 整数値16961(内部表現＝4241H)を２バイトの文字コード"AB"に変換します．

プログラム例

ディスク

```
list
100 ' MKI$/MKS$/MKD$ sample
110 OPEN "2:rdata" AS #1
120 FIELD #1,2 AS D1$,4 AS D2$,8 AS D3$
130 A%=123:B=8534.13:C#=3.141592654000002#
140 LSET D1$=MKI$(A%)
150 LSET D2$=MKS$(B)
160 LSET D3$=MKD$(C#)
170 PUT #1,1
180 CLOSE
190 END
Ok
```

• 130行にあるデータを，"rdata"というランダムファイルに書き込みます．

• 140行から160行で，整数型を２バイトの文字列，単精度実数型を４バイトの文字列，倍精度実数型を８バイトの文字列に型変換をしています．

参照：CVI/CVS/CVD

MON

モニタ：monitor

機能 機械語モニタに制御を移す

書式 MON

解説

・BASICモードから機械語モニタに制御を移すためのコマンドです．

・機械語モニタがコマンドレベルに入ると，"＊"というプロンプトを表示し，コマンド待ちの状態になります．

・CTRL ＋ B でBASICモードに戻ります．

・機械語モニタの使い方については，PC-8001MKⅡSRユーザーズガイドを参照してください．

例 MON

・機械語モニタに制御を移します．

実 行 例

```
mon⏎
* CTRL+B
Ok
```

・BASICモードから機械語モニタに制御を移します．アスタリスク(＊)が表示されたあと CTRL ＋ B を押して，BASICのコマンドレベルに戻ります．

MOTOR

モータ：motor

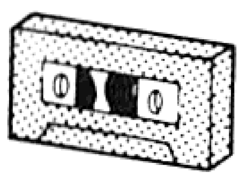
機能 カセットテープレコーダのモータを制御する

書式 MOTOR〔〈スイッチ〉〕

解説
- 〈スイッチ〉を0にすればモータはOFFに，0以外の値にすればONの状態になります．
- 〈スイッチ〉を省略した場合，モータがOFFの状態ならONに，ONの状態ならOFFになります．

例 MOTOR 1
- カセットテープレコーダのモータをONにします．

プログラム例


カセットテープ

```
list↵
100 ' MOTOR sample
110 MOTOR 1
120 PRINT "Rewind and set the tape."
130 INPUT "Are you ready (y/n)";A$
140 IF LEFT$(A$,1)<>"y" THEN END
150 PRINT:MOTOR 0
160 PRINT "Push PLAY button."
170 INPUT "Are you ready (y/n)";A$
180 IF LEFT$(A$,1)<>"y" THEN END
190 MOTOR
200 END
Ok
run↵
Rewind and set the tape.
Are you ready (y/n)? y↵

Push PLAY button.
Are you ready (y/n)? y↵
Ok
```

- 110行で，カセットテープレコーダのモータをONの状態にして，カセットテープを巻き戻すことができるようにします．130行で"y"を読み込んだ場合，カセットテープがセットされ，巻き戻されたと判断してひとまずモータをOFFにしておきます．次に，170行で"y"を読み込んだ場合，PLAYボタンが押されていると判断してモータをONにします．

NAME

ディスク

ネーム：name

機能 フロッピィディスク上のファイルの名前を替える

書式 NAME〈旧ファイル名〉AS〈新ファイル名〉

解説 •〈旧ファイル名〉で表されているフロッピィディスク上のファイルを〈新ファイル名〉に変更します．

• ドライブ番号は，ファイル名の前に "n:" を付けて指定します（例："2:sample"）．省略した場合は，ドライブ1が指定されたとみなされます．

• NAMEコマンドによりファイル名の変更を行うとき，そのファイルは閉じられた状態でなければなりません．

例 NAME " sumple " AS " sample "

• ドライブ1に入っているフロッピィディスク上の "sumple" というファイル名を，"sample" に替えます．

プログラム例

ディスク

```
list
100 ' NAME sample
110 OPEN "2:oldtest" FOR OUTPUT AS #1
120 FOR J=1 TO 10
130   PRINT #1,J
140 NEXT J:CLOSE
150 FILES 2:PRINT
160 NAME "2:oldtest" AS "2:newtest"
170 FILES 2
180 END
Ok
run
data1       1    data2       1    data4       1    data5       1    data6       1
data7       1    data8       1    data9       1    addres s    1    rdata       1
animal      1    test   .nsr 1    test2 .nsr  1    test3  nsr  1    test4 .nsr  1
fdata       1    usr    .    1    oldtes t    1

data1       1    data2       1    data4       1    data5       1    data6       1
data7       1    data8       1    data9       1    addres s    1    rdata       1
animal      1    test   .nsr 1    test2 .nsr  1    test3  nsr  1    test4 .nsr  1
fdata       1    usr    .    1    newtes t    1
Ok
```

• ドライブ2のフロッピィディスクに "oldtest" というファイルを作成し，"newtest" というファイル名に変更します．

参照：KILL

NEW

ニュー・new

機能 メモリ上にあるプログラムを消去する

書式 NEW

解説

• メモリ上にあるプログラムを消去します．同時に，すべての変数を初期化します．

• NEWコマンドはコマンドレベルにあるとき，新しいプログラムを入力する前に実行します．NEWコマンドの実行が終わるとコマンドレベルに戻ります．
NEWコマンドはオープンされているファイルがある場合，それを自動的に閉じます．

例 NEW

• プログラムを消去します．

プログラム例

```
list
100 ' NEW sample
110 ' NEW initializes the program.
120 PRINT "Don't use NEW in a program like this!"
130 PRINT "Program is deleted."
140 NEW
150 END
Ok
run
Don't use NEW in a program like this!
Program is deleted.
Ok
list
Ok
```

• 120行と130行でメッセージを出力して，140行でプログラムを消しています．

OCT$

オクタル・ダラー：octal $

機　能　10進数を８進数の文字列に変換した値を返す

書　式　OCT$(〈数式〉)

解　説

• 〈数式〉の値を８進数に変換して，その文字列を返します．

• 〈数式〉の値の範囲は，−32768～65535です．〈数式〉の値に小数点以下の値が含まれる場合は，小数点以下を切り捨てたあと変換します．

例

OCT$(−100)　返される値　" 177634 "

OCT$(100)　返される値　" 144 "

OCT$(65535)　返される値　" 177777 "

プログラム例

```
list
100 ' OCT$ sample
110 FOR I=0 TO 16
120    DE$=RIGHT$("   "+STR$(I),3)
130    OC$=RIGHT$("   "+OCT$(I),3)
140    HE$=RIGHT$("   "+HEX$(I),3)
150    PRINT "DECIMAL:";DE$,"OCTAL:";OC$,"HEX:";HE$
160 NEXT I
170 END
Ok
run
DECIMAL:  0   OCTAL:  0     HEX:  0
DECIMAL:  1   OCTAL:  1     HEX:  1
DECIMAL:  2   OCTAL:  2     HEX:  2
DECIMAL:  3   OCTAL:  3     HEX:  3
DECIMAL:  4   OCTAL:  4     HEX:  4
DECIMAL:  5   OCTAL:  5     HEX:  5
DECIMAL:  6   OCTAL:  6     HEX:  6
DECIMAL:  7   OCTAL:  7     HEX:  7
DECIMAL:  8   OCTAL: 10     HEX:  8
DECIMAL:  9   OCTAL: 11     HEX:  9
DECIMAL: 10   OCTAL: 12     HEX:  A
DECIMAL: 11   OCTAL: 13     HEX:  B
DECIMAL: 12   OCTAL: 14     HEX:  C
DECIMAL: 13   OCTAL: 15     HEX:  D
DECIMAL: 14   OCTAL: 16     HEX:  E
DECIMAL: 15   OCTAL: 17     HEX:  F
DECIMAL: 16   OCTAL: 20     HEX: 10
Ok
```

• 0から16までの10進数を，８進数と16進数で表すプログラムです．

• 120行から140行でRIGHT$関数を使っているのは，空白を含め，３文字にして桁をそろえて出力するためです．

参照：HEX$

ON ERROR GOTO

オン・エラー・ゴートゥー：on error goto

機能 エラートラップを可能にし，エラー処理ルーチンの開始行を定義する

書式 ON ERROR GOTO〈行番号〉

解説

• エラートラップ機能が可能になるとエラーが検出されたとき，指定されたエラー処理ルーチンへプログラムの制御が移ります．

•〈行番号〉の行が存在しなければ，"Undefined line number"（未定義行）のエラーになります．

• エラートラップ機能を無効にするには，ON ERROR GOTO 0を実行してください．

• エラー処理サブルーチンの実行中にエラーが起きた場合には，それに対応するエラーメッセージが表示され，実行は停止します．エラー処理サブルーチンの中では，エラートラップは起こりません．

• エラー処理後，プログラムの実行を再開するにはRESUME文を使います．

• エラー処理ルーチンにON ERROR GOTO 0の文がある場合には，BASICはエラートラップの原因となったエラーのエラーメッセージを表示し，プログラムの実行を停止します．すべてのエラー処理ルーチンにおいて，エラー回復処理を行わないエラーをトラップした場合には，ON ERROR GOTO 0を実行することをお勧めします．

例 ON ERROR GOTO 1000

• エラートラップを可能にします．これを実行した後，エラーが発生したら1000行から実行します．

プログラム例

```
list↵
100 ' ON ERROR GOTO sample
110 ON ERROR GOTO 160
120 INPUT "x :";X:INPUT "y :";Y
130 IF X=0 AND Y<0 THEN ERROR 250
140 Z=X^Y:PRINT "  y":PRINT " X=";Z
150 GOTO 120
160 IF ERR=250 THEN PRINT " ﾃｲｷﾞ ｻﾚﾏｾﾝ｡"
170 IF ERR=6   THEN PRINT " ｵｰﾊﾞｰ ﾌﾛｰ ﾃﾞｽ｡"
180 RESUME 120
Ok
run↵
x :? 2↵
y :? 10↵
  y
 X= 1024
x :? 100↵
```

```
y :? 100⏎
 オーハ゛ー フロー テ゛ス。
x :? 0⏎
y :? -2⏎
 テイキ゛ サレマセン。
x :? STOP
Break in 120
Ok
```

• 2つの実数XとYを読み込んで，X^Yを出力します．

• 110行で，エラーが発生したら160行から実行するように設定しています．

• Xに0，Yに負の数を入力した場合，130行で250番のエラーを発生させ，160行でメッセージを出します．その他のエラーで160行に実行が移ったときは，オーバーフローだけはメッセージを出力しますが，それ以外は何もしないで120行に戻ります．

参照：RESUME, GOTO

ON...GOSUB/ON...GOTO

オン・ゴーサブ/オン・ゴートゥー：on...go to subroutine/on...go to

機能 指定されたいくつかの行に分岐する

書式 ON〈数式〉GOSUB〈行番号〉〔,〈行番号〉...〕

ON〈数式〉GOTO〈行番号〉〔,〈行番号〉...〕

解説 •〈数式〉の値がプログラムのどの行番号の行に分岐するかを決定します．〈行番号〉の並びは1から始まる数に対応します．たとえば，〈数式〉の値が3であったなら，行番号のリストの3番目の行が分岐点となります．

•〈数式〉の値の範囲は0～255です．値が0または行番号リストの個数より大きくなった場合には，次の行へプログラムの制御が移ります．

例 ON N GOTO 100,200,300

•Nの値によって，100行(N=1),200行(N=2),300行(N=3),次の行(それ以外)へ分岐します．

プログラム例

```
list
100 ' ON..GOSUB/ON..GOTO sample
110 INPUT "N=";N
120 SG=SGN(N)+1
130 ON SG GOTO 150,160
140 PRINT "MINUS!":GOTO 110
150 PRINT "ZERO!":GOTO 110
160 PRINT "PLUS!":GOTO 110
Ok
run
N=? 0
ZERO!
N=? 51
PLUS!
N=? -24
MINUS!
N=? STOP
Break in 110
Ok
```

•単精度実数を読み込んで，その数が正か負かあるいは0かを出力するプログラムです．

•120行で変数SGの値は，Nが負のときは0，Nが0のときは1，Nが正のときは2になります．130行で，それぞれのメッセージを出力する行に分岐します．

参照：GOSUB, GOTO

OPEN

ディスク

オープン：open

機能

フロッピィディスク上のファイルをオープンする

書式

OPEN〈ファイル名〉〔FOR〈モード〉〕AS〔#〕〈ファイル番号〉

解説

• 〈ファイル名〉で指定されたフロッピィディスク上のファイルを，指定された〈ファイル番号〉でオープンします．以後，オープンされたファイルへの入出力は，〈ファイル番号〉を指示することにより行います．

• 〈モード〉は，ファイルのアクセス方法を指定します．モードには次の4種類があります．

INPUT ――――	既存のシーケンシャルファイルから入力を行うことを指示する
OUTPUT ――――	新しくシーケンシャルファイルを作り出力を行うことを指示する
APPEND ――――	既存のシーケンシャルファイルの終わりから追加を行うことを指示する
省略したとき ――	FOR〈モード〉が省略されると，ランダムファイルに対して入出力を行うことを指示する

• INPUT，APPENDモードでは指定されたファイルが存在しないと，"File not found"エラーとなります．OUTPUTモードでは，常に指定された名前のファイルを新しく作り，同一名のファイルがあった場合そのファイルは削除されます．ランダムアクセスでファイルが存在しない場合には新たにファイルが作られます．

• 〈ファイル番号〉は1～15の値を用いることができますが，これは，システム起動時に"How many files(0−15)?"で指定したファイルの数を超えてはいけません．

• OPEN文は以後の入出力の際に用いるバッファ領域を確保し，ファイルが閉じられるまでの間そのバッファは，指定したファイルへの入出力操作専用に使われます．このバッファはファイル番号と同一の番号で参照されます．

例

```
OPEN " sample " FOR INPUT AS #1
```

• "sample"というファイル名のシーケンシャルファイルを入力ファイルとしてファイル番号1で開きます．

プログラム例

```
list↵
100 ' OPEN sample
110 OPEN "2:data9" FOR INPUT AS #1
120 OPEN "2:newdata9" FOR OUTPUT AS #2
130 IF EOF(1) THEN 170
140   INPUT #1,A$
150   PRINT #2,A$
160 GOTO 130
170 CLOSE
180 END
Ok
```

• ドライブ2のフロッピィディスクに入っている "data9"というシーケンシャルファイルの内容を，ドライブ2に入っているフロッピィディスク上の "newdata9"というシーケンシャルファイルにコピーします．

参照：CLOSE, VARPTR, FIELD

OUT

アウト：out

機能　出力ポートに１バイトのデータを送る

書式　OUT〈I/Oアドレス〉,〈数式〉

解説
- 〈I/Oアドレス〉は出力ポートの番号,〈数式〉は出力する１バイトのデータです.
- 〈I/Oアドレス〉,〈数式〉は共に０～255の整数値で指定します.

例　OUT &H54,&H03

- カラーパレット０に，３(紫)を指定します.（54Hポートは，カラーパレットを制御するポートです.）

プログラム例

```
list
100 ' OUT sample
110 SCREEN 0,0:COLOR=(7,7)
120 CIRCLE(320,100),50,7
130 PAINT(320,100),7,7
140 FOR J=1 TO 100
150   FOR I=1 TO 7
160     OUT &H5B,I
170     FOR K=0 TO 100:NEXT K
180   NEXT I
190 NEXT J
200 END
Ok
```

- COLOR＝文のプログラム例と同じ動作をします.
- 160行で，OUT文を使って直接パレットを書き換えています.

注意：• OUT文は十分なハードウェアの知識を持たずに使うと，BASICが正常に動作しなくなることがあります.

参照：INP

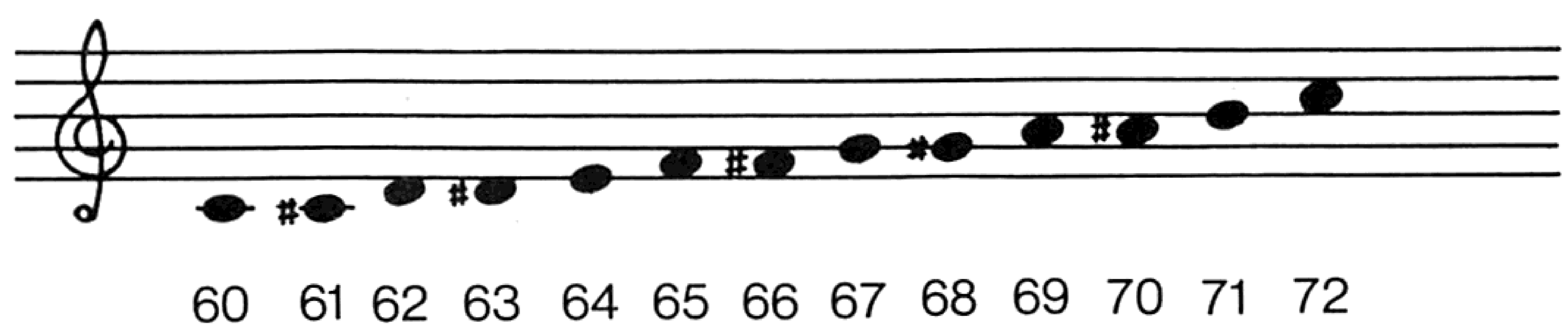
OUTM

アウト・エム：output midi

機能　MIDIポートへ１バイトのデータを送る

書式　OUTM〈数式〉〔,〈数式〉...〕

解説
・このステートメントは，本体にミュージックインタフェースボード（PC-8801-10）が接続してあるときに使用します．
・音程は０～127の範囲で指定します．



半音上がるごとに＋１されます．第４オクターブのＣ（ド）の音はキーナンバが60になります
・音量は０～127の範囲で設定します．PLAY文の"V"の値とOUTM文の音量の対応は次のようになります．

PLAY文の"V"の値	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OUTM文の音量	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

・詳しくはミュージックインタフェースボードに添付されているユーザーズマニュアルを参照してください．

例　OUTM 60
・MIDIポートへ１バイトのデータ（60）を送り出します．

プログラム例

このプログラムを実行する場合には，別売のミュージックインタフェースボードが必要です．

```
list
100 ' OUTM sample
110 OUTM &H90
120 OUTM 60
130 OUTM 64
140 FOR W=0 TO 500:NEXT W
150 OUTM &H80
160 OUTM 60
170 OUTM 0
```

```
180 END
Ok
```

•第4オクターブのCの音を出力し，その後音を消します．

•110行から130行で，ステータス&H90，音程60(第4オクターブのCの音)および音量64(PLAY文の"V8"と同音量)をMIDIポートに送ります．

•150行から170行で，ステータス&H80，音程60，音量0を送ります．

注意： •本体にミュージックインタフェースボードを接続して使用する場合，ミュージックインタフェースボードのジャンパスイッチのジャンパコネクタは取り外してください．

〔1〕PAINT

ペイント：paint

機能 指定された境界色で囲まれた領域を，指定された色で塗る

書式

```
PAINT |(gx,gy)        |〔,〈領域色〉〕〔,〈境界色〉〕
      | STEP(sx,sy)   |
```

解説 • グラフィック座標(gx, gy)で指定された点を含む〈境界色〉で囲まれた領域を，指定された〈領域色〉で塗りつぶします.

• 〈領域色〉，〈境界色〉はともにパレット番号で指定します. 〈領域色〉が省略された場合には，COLOR文で指定された〈フォアグラウンドカラー〉が用いられます. 〈境界色〉が省略された場合には〈領域色〉と同じパレット番号の色が用いられます.

• (gx, gy)は塗り始める座標を指定します. もしこの点が，すでに指定された境界色と同じ色であった場合には，PAINT文は何の画面操作も行いません. PAINT文はビューポート内でのみ働きますので，ビューポートの境界はPAINT文の動作の境界とみなされます.

• 塗り始めの座標がビューポートの範囲外にある場合は，"Illegal function call"エラーとなります.

例 PAINT(50,50),4,7

• パレット番号7の色で囲まれた領域を，グラフィック座標(50，50)を基点として，パレット番号4の色で塗りつぶします.

プログラム例

```
list
100 ' PAINT sample
110 SCREEN 0,0:CLS 3
120 X=INT(RND(1)*639):Y=INT(RND(1)*199)
130 R=RND(1)*50+1:C=RND(1)*6+1
140 CIRCLE(X,Y),R,C
150 PAINT(X,Y),C
160 GOTO 120
Ok
```

• いろいろな円を描き，その円の中を円を描いた色で塗りつぶします.

• 120行および130行で円の中心座標，半径，色を乱数を使って求めています.

注意：• 複雑な図形を塗りつぶすとき，"Out of memory"エラーとなることがあります.

参照：COLOR, VIEW

(2) PAINT

ディスク

ペイント：paint

機能　指定された境界色で囲まれた領域を，指定されたタイルパターンで埋める

書式

```
PAINT | (gx,gy)        | [,〈タイルストリング〉][,〈境界色〉]
      | STEP(sx,sy)    |
```

解説

・グラフィック座標(gx, gy)で指定された点を含む〈境界色〉で囲まれた領域を，指定された模様で埋めます．この動作はタイリングと呼ばれ，模様のついたタイルを領域内に敷きつめたり，中間色で領域内を塗ったような効果を出すことができます．

・〈タイルストリング〉は，タイリングに用いられる基本タイルの模様と大きさを決める文字列です．タイルの大きさは，横方向は8ドット分と決められていますが，縦方向の長さは，タイルストリングの長さで指定することができます．縦方向がNドットのタイルを指定するためには，白黒モードでN文字，カラーモードで3N文字の長さを必要とします．タイルストリングとして用いられる文字の長さは最大64文字までで，それ以上は無視されます．

・タイルの模様は，〈タイルストリング〉に対応するキャラクタコードのビットパターン(2進数表現)により表現されます．〈タイルストリング〉の指定の方法は，カラーモードと白黒モードとで異なります．白黒モードでは，文字を横8ビットの線に対応させて指定します．文字コードは8ドットのうちどのドットを描き，どのドットを消去するかを決定し，2進数表現で1のビットに対応するドットは描かれ，0に対応するドットは消去されます．〈タイルストリング〉がN文字の長さであれば，そのストリングの表す模様は，このようにして決定される横8ドットのパターンを縦にN文字のパターン分だけ並べたものになります．したがって，タイルストリングを構成する文字がすべて同じ文字であったり，1文字だけであった場合には，縦線模様になります．

例)　CHR$(&HAA)＋CHR$(&H55)

16進数表現	2進数表現	ドットパターン
&HAA	10101010	●○●○●○●○
&H55	01010101	○●○●○●○●

この例は，2文字からなる簡単なタイルストリングの例です．このパターンを基本タイルとして指定された領域を埋めれば，細かい市松模様になります．

・カラーモードでも白黒モードと同じように，模様はタイルストリングに対応す

るビットパターンによって決定されますが，白黒モードとは異なり３文字で横８ドットが構成されます．ストリング中の文字は先頭から，青，赤，緑のドットパターンを決定していきます．

•カラーモードのとき，ストリングの長さに余りがあった場合には，残りの文字は無視されます．また，３文字に満たない場合には，"Illegal function call" エラーとなります．

例）　CHR$(&H55)＋CHR$(&H33)＋CHR$(&H0F)

16進数表現	2進数表現
&H55	01010101
&H33	00110011
&H0F	00001111
	↓↓↓↓↓↓↓↓
パレット番号——→	01234567

上の例は，３文字からなるタイルストリングの例で，８×１ドットのラインパターンを作ることができます．この３文字を並べた状態で各ビットを上記のように"たて"に読むと，左から順に0，1，2……７となります．これが各ドットのパレット番号になり，この例でタイリングを行うと，各ドットごとに異った色を持つ８ドットのラインで領域が埋められます．また，COLOR＝文によってパレットへのカラーコードの割り付けを変化させると各ドット単位で色を変えることができます

例　PAINT(50,50),"33フ",7

•パレット番号７の色で囲まれた領域を，グラフィック座標(50，50)を基点として，タイルストリング(CHR$(&H33)＋CHR$(&H33)＋CHR$(&HCC))で埋めます．カラーモードで実行すると，緑と紫の縦縞模様となります．

プログラム例

ディスク

```
list
100 ' PAINT sample
110 SCREEN 0,0:CLS 3
120 FOR Y=0 TO 3
130   FOR X=0 TO 7
140     LINE(X*80,Y*50)-STEP(80,50),7,B
150     B$=CHR$(RND(1)*255):R$=CHR$(RND(1)*255)
160     G$=CHR$(RND(1)*255)
170     TILE$=B$+R$+G$
180     PAINT(X*80+40,Y*50+25),TILE$,7
190   NEXT X
200 NEXT Y
210 END
Ok
```

• タイルパターンを32個表示させます.

• 150行から160行で，青，赤，緑，それぞれのドットパターンを作っています. RND(1)＊255で，16進の0HからFFHまでの乱数が得られます.

• 170行で文字型変数TILE$に，150行および160行で作ったタイルストリングを代入しています.

• 140行で白い四角形を描き，180行でその中をタイルパターンで塗りつぶします.

注意： • 既にタイリングされている領域を異なるパターンでタイリングしようとすると，時間がかかったり，メモリが不足して"Out of memory"エラーとなることがあります.

PEEK

ピーク：peek

機能 メモリ上の指定された番地の内容を返す

書式 PEEK(〈番地〉)

解説
- 〈番地〉によって指定されたメモリ番地の内容を0〜255の値で返します．
- 〈番地〉の範囲は0〜65535です．小数点が含まれる場合は小数点以下が切り捨てられます．

例 PEEK(&H9000) 返される値 (9000H番地の内容)

プログラム例

```
list
100 ' PEEK sample
110 INPUT "start address(hex)";SA$
120 SA=VAL("&h"+SA$)
130 INPUT "end   address(hex)";EA$
140 EA=VAL("&h"+EA$)
150 CO=1
160 PRINT RIGHT$("000"+HEX$(SA),4);": ";
170 FOR I=SA TO SA+15
180   DA=PEEK(I)
190   PRINT RIGHT$("0"+HEX$(DA),2);" ";
200 NEXT I
210 SA=SA+16:PRINT
220 IF SA<=EA THEN 160
230 END
Ok
run
start address(hex)? 8000
end   address(hex)? 8050
8000: C3 B8 88 C3 5A 8A C3 C2 8B C3 FA 92 C3 AD 92 C3
8010: 3C 89 C3 29 02 C3 F0 80 C3 00 81 C3 32 05 C3 3E
8020: 05 C3 4B 02 C3 5F 02 C3 74 05 C3 EA 05 C3 FB 02
8030: C3 3E 80 C3 16 81 3E FF D3 E4 32 5B E6 C9 F5 3A
8040: 86 EC FE 01 28 02 F1 C9 E5 C5 3A 8F EC B7 20 1B
8050: 3A 47 EC 4F 3E 05 ED 79 21 02 00 CD DF 04 3E 0F
Ok
```

- 指定された範囲のメモリの内容を16進数で表示します．
- 160行では，16進数を4桁に，190行では2桁にそろえて出力するために，RIGHT$関数を使って，それぞれの桁より少ない値のときは左側に0を補って出力します．

参照：POKE

PLAY

プレイ：play

機能 音楽を発生する

書式 PLAY〔#〈音源〉,〕〔〈文字列1〉〕〔,〈文字列2〉〕〔,〈文字列3〉〕〔,〈文字列4〕〔,〈文字列5〉〕〔,〈文字列6〉〕

解説
- 6和音まで演奏することができます.
- 〈文字列1〉,〈文字列2〉,〈文字列3〉はチャネル1，2，3に,〈文字列4〉,〈文字列5〉,〈文字列6〉はチャネル4，5，6に対応します.
- 〈音源〉は0〜4の値で指定します. この〈音源〉をプログラミング時に変数にしておくと，切り替えが容易になります.

#0： 別売のミュージックインタフェースボードのSSG音源を指定します. このボードにはPSGが2個実装されており，チャネル1，2，3でPSG1の各SSG音源を，チャネル4，5，6でPSG2の各SSG音源を制御します.

PSG1			PSG2		
SSG音源			SSG音源		
チャネル1	チャネル2	チャネル3	チャネル4	チャネル5	チャネル6

#1： 別売のミュージックインタフェースボードのMIDIインタフェースを指定します. この場合，MIDI仕様のシンセサイザーが必要です. 接続されたキーボードによりチャネルの最大数は異なりますが，PLAY文では最大が6となります.

#2： 省略時の場合の初期値です. 内蔵シンセサイザーIC(OPN)のFM音源，SSG音源を指定します. チャネル1，2，3が各FM音源を，チャネル4，5，6が各SSG音源を制御します.

OPN					
FM音源			SSG音源		
チャネル1	チャネル2	チャネル3	チャネル4	チャネル5	チャネル6

#3： OPNを効果音モードに切り替えます. PLAY文中のYコマンドに

よってOPNのレジスタの内容を書き換えることにより音を出します.

#4： OPNをCSM(サイン波)モードに切り替えます．PLAY文中のYコマンドによってOPNのレジスタの内容を書き換えることにより音を出します.

注意：通常，音源のモードは#0, #1，あるいは#2でお使いください．#3, #4で使用する場合にはOPNの構造を充分に理解することが必要です(資料4 FM音源とシンセサイザーIC参照).

• 各チャネルの文字列は，MML(Music Macro Language)といい，次のような意味を持ちます.

文　　字	意　　味	デフォルト値
Mx(SSG音源のみ)	エンベロープ周期の設定.(1≦x≦65535)	M255
Sx(SSG音源のみ)	エンベロープ形状の設定.(0≦x≦15)	S1
Vx	音量の設定.(0≦x≦15)	V8
Lx	長さの設定.(1≦x≦64)	L4
Qx	音の長さの割合.(1≦x≦8)	Q8
Ox	オクターブの設定.(1≦x≦8)	O4
>	オクターブを1つ上げる.	
<	オクターブを1つ下げる.	
Nx	xで指定された高さの音を発生する.(0≦x≦96)	
Tx	テンポの設定.(32≦x≦255)	T120
Rx	休符の設定.(1≦x≦64)	R4
+(#)	音を半音上げる.	
−	音を半音下げる.	
.	音符の長さや休符の長さを1.5倍にする.	
&	タイ．前後の音を継なぐ.	
{ }x	連符．指定された長さのX分音符を{ }の中の音程の個数で等分にした音を発生する.	
@x(FM音源のみ)	xで指定された音色番号に切り替える.(音色番号表参照)	
Yr, d(OPNのみ)	OPNのレジスタrの内容をdにする.	
Zd(MIDIのみ)	MIDIにデータdを送る.	
@Vx(FM音源, MIDIのみ)	音量を細かく調整する.(0≦x≦127)	
@Wx	xで指定された長さだけ状態を維持する.(1≦x≦64)	@W4

初期値を持っているパラメータは，省略したり，PLAY文実行中 **STOP** キーで止めたり，"Illegal function call" エラー等でストップすると，初期値に戻ります.

1. 音程(Cx, Dx, Ex, Fx, Gx, Ax, Bx)

音程はC〜Bで表し，パラメータxは音の長さを表します．その対応は次

のとおりです．パラメータｘが省略されたときはＬコマンドの値が設定されたものとみなします．

文字	C	D	E	F	G	A	B
音程	ド	レ	ミ	ファ	ソ	ラ	シ

x	1	2	4	8	16	32	64
長さ	全音符	2分音符	4分音符	8分音符	16分音符	32分音符	64分音符

2. タイ(&)

前後の音をつなぎます．前後の音程が異なる場合と後に音程以外の文字を指定すると，前の音だけ "Q8" を指定した場合と同じ効果になります．

3. シャープ(＋，#)

音を半音上げます．ただし，"B＋(B#)" としても1オクターブ上の "C" にはなりません．同じオクターブの "C" になります．

4. フラット(－)

音を半音下げます．ただし，"C－" としても1オクターブ下の "B" にはなりません．同じオクターブの "B" になります．

5. 符点(．)

音符や休符の長さを1.5倍にします．

6. 連符({ }x)

$\frac{\text{指定された長さx}}{\text{\{ \}の中の音符の数}}$ によって等分された長さで音を発生します．等分した際，1音の長さが64音符より短くなる場合はエラーになります．また{ }内で音符の長さを変えるコマンドを指定した場合，連符にはなりません．

7. Vコマンド(Vx)

音量の設定をします．パラメータｘは0～15の範囲の値で設定し，初期値は "V8" です．"V15" が最大音量になります．

8. Lコマンド(Lx)

音の長さを設定します．パラメータｘは1～64の範囲の値で，初期値は "L4" です．パラメータｘと音符の対応は，音程のパラメータｘと同様です．

9. Qコマンド(Qx)

1音中の音の長さの割り合いを設定します．パラメータｘと長さの割り合

いの対応は次のとおりです．

パラメータ	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
音の長さの割合	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8

初期値は，1音中の音の長さがすべて出力される"Q8"が指定されます．

MIDIを使用している場合に，"Q8"を指定して同じ高さの音を続けて演奏すると，シンセサイザーの機種や音色によって，第2音以降の音が出ないことがあります．そのときはパラメータxの値を小さく設定してください．

10. Oコマンド(Ox)

オクターブの設定をします．パラメータxは1〜8の値で，初期値は"O4"です．


O4

なお，Oコマンドとは別に現在のオクターブから1オクターブ上げる"＞"と1オクターブ下げる"＜"があります．

11. Rコマンド(Rx)

休符の設定をします．パラメータxは1〜64の値で初期値は"R4"です．パラメータxと休符の対応は次のとおりです．

x	1	2	4	8	16	32	64
長さ	全休符	2分休符	4分休符	8分休符	16分休符	32分休符	64分休符

パラメータxなしのRコマンドを設定した場合，その休符の長さは，Lコマンドが設定されていた場合はその長さと同じになります．また，各チャネルの音を発生する長さが違った場合，長さの短い方の残りの部分はすべてR(レスト)とみなされます．

12. Tコマンド(Tx)

テンポの設定をします．パラメータxは32〜255の値で，初期値は"T120"です．"T120"とは1分間に4分音符が120回演奏できる速さで，楽譜によく書かれている♩=120に当たります．

TコマンドとLコマンドのパラメータxを両方とも大きな値にする(速い

テンポで短い音符を演奏するなど)と，時間の計算の誤差により，実際の速さや長さと異なって聞こえる場合があります．

13. Nコマンド(Nx)

パラメータxで指定された高さの音を発生します．パラメータxは0〜96の値です．なお，FM音源の場合，"N96"は"N0"に等しくなります．その他は次のとおりです．

Nコマンドパラメータと音程の対応

	O1	O2	O3	O4	O5	O6	O7	O8	O9
C	0	12	24	36	48	60	72	84	96
C+(D−)	1	13	25	37	49	61	73	85	
D	2	14	26	38	50	62	74	86	
D+(E−)	3	15	27	39	51	63	75	87	
E	4	16	28	40	52	64	76	88	
F	5	17	29	41	53	65	77	89	
F+(G−)	6	18	30	42	54	66	78	90	
G	7	19	31	43	55	67	79	91	
G+(A−)	8	20	32	44	56	68	80	92	
A	9	21	33	45	57	69	81	93	
A+(B−)	10	22	34	46	58	70	82	94	
B	11	23	35	47	59	71	83	95	

14. @Wコマンド(@Wx)

パラメータxで指定された長さだけ，現在の状態(Key-on/off)を維持します．Key-onの状態を続けるには，YコマンドでKey-onした後だけ有効です．音程(C〜B)はKey-offしてから次のMMLコマンドに実行を移しますので，Rコマンドと同じ効果しか得られません．パラメータxが省略された場合は，Lコマンドの値が設定されたものと見なします．

15. @コマンド(@x)

FM音源をパラメータxで指定された音色番号の音色に切り替えます．音色番号と音色の対応は音色番号表を参照してください．また，@コマンドを指定した場合はパラメータxを省略することができませんので，必ずパラメータを設定してください．ただし，@コマンドが実行されるまでは音色番号0が指定されているとみなされます．

音色番号表

音色番号：音色名	音色番号：音色名
0：Default VOICE	31：EBASS 2
1：BRASS 2	32：EORGAN 1
2：STRING 2	33：EORGAN 2
3：EPIANO 3	34：PORGAN 1
4：EBASS 1	35：PORGAN 2
5：EORGAN 1	36：FLUTE
6：PORGAN 1	37：PICCOLO
7：FLUTE	38：OBOE
8：OBOE	39：CLARINET
9：CLARINET	40：GROCKEN
10：VIBRPHN	41：VIBRPHN
11：HARPSIC	42：XYLOPHN
12：BELL	43：KOTO
13：PIANO	44：ZITAR
14：MUSHI	45：CLAV
15：DESCENT	46：HARPSIC
16：UFO	47：BELL
17：GRANPRI	48：HARP
18：LASER 1	49：BELL/BRASS
19：LASER 2	50：HARMONICA
20：SIN WAVE	51：STEEL DRUM
21：BRASS 1	52：TIMPANI
22：BRASS 2	53：TRAIN
23：TRUMPET	54：AMBULAN
24：STRING 1	55：TWEET
25：STRING 2	56：RAIN DROP
26：EPIANO 1	57：HORN
27：EPIANO 2	58：SNARE DRUM
28：EPIANO 3	59：COW BELL
29：GUITAR	60：PERC 1
30：EBASS 1	61：PERC 2

音色番号1から12までは，他のPCシリーズとの互換性を保つために用意されています．

0：音色番号が全く指定されなかったとき，ハープシコードが割り当てられます．

1,21,22：ブラス系の音

2,24,25：ストリングス系の音

3,26,27,28：エレクトリックピアノ

4,30,31：エレクトリックベース

5,32,33：エレクトリックオルガン

6,34,35：パイプオルガン

10,41：ビブラホン

11,46：ハープシコード

14：虫の鳴き声

15：高空から降下する音

16：UFOが遠ざかる音

17：レーシングカーのエンジン音

18,19：レーザーガン

20：チューニング用の正弦波

40：グロッケン

42：シロホン

44：ツィター

45：クラビネット

49：スタッカートでベル，ロングトーンでブラスの音

53：列車の警笛

54：救急車

55：小鳥のさえずり

56：雨の落ちる音

58：スネアドラム

60,61：パーカッション

16. @Vコマンド(@Vx)

FM音源とMIDIの音量を細かく設定します．パラメータxは0～127の値です．なお，このコマンドはVコマンドと関係なく設定されます．"V15"と設定してあっても，"@V0"とすれば音量は0になります．"@V127"が最大音量です．Vコマンドと@Vコマンドのパラメータxの対応は次のようになります．

Vコマンド	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
@Vコマンド	85	87	90	93	95	98	101	103	106	109	111	114	117	119	122	125

@Vコマンドを指定した場合はパラメータxを省略できませんので，必ずパラメータxを指定してください．

17. Yコマンド(Yr, d)

OPNのレジスタrの内容をdに設定します．OPNレジスタ及び設定値については，資料4　FM音源とシンセサイザーICを参照してください．

18. Zコマンド(Zd)

MIDIにデータdを送ります．

19. Mコマンド(Mx)

SSG音源のエンベロープ周期の設定をします．パラメータxは1〜65535の範囲の値で設定し，初期値は"M255"です．パラメータxは次の式によって求めることができます．

$$x = fclock * T/256 \quad \begin{pmatrix} \text{fclock: 基本周波数(1996800Hz)} \\ \text{T: 周期(秒)} \end{pmatrix}$$

20. Sコマンド(Sx)

SSG音源のエンベロープ形状の設定をします．パラメータxは0〜15の範囲の値で設定し，初期値は"S1"です．パラメータxとエンベロープ形状の関係はSOUND文を参照してください．

• MMLの各パラメータには変数を使うことができます．MMLのあとに"＝変数名；"で指定します．

例)
```
10 VL=15:RG=40:DT=240
20 PLAY #2,"V=VL;CDE"
30 PLAY #2,"Y=RG;,=DT;@W1"
```

また，音程のあとに長さを指定する場合にも変数を使うことができます(ただし，ミュージックインタフェースボードに添付のカセットテープの中に入っているプログラムを実行して拡張されたBASICでは使用できません)．

例)
```
10 LN=2
20 PLAY #2,"L4C=LN;DE"
```

例 PLAY #2,"c","e","g"

• 内蔵のOPNのFM音源で"ドミソの和音を出します．なお，この場合，"#2,"は省略できます．

プログラム例

```
list
100 ' PLAY sample
110 S=2:T=11
120 PLAY #S,"@=T;T180O5l4"
130 A$="GB-2O6CD4.E8DC2O5AF4.G8A"
140 B$="O6F2F8.R16F4.E8DC2O5AF4.G8A"
150 PLAY #S,A$+"B-2GG4.F+8GA2F+D2"
```

```
160 PLAY #S,A$+"B-4.A8GF+4.E8F+G2G8.R16G2."
170 PLAY #S,B$+"B-2GG4.F+8GA2F+D2."
180 PLAY #S,B$+"B-4.A8GF+4.E8F+G2G8.R16G2."
190 END
Ok
```

• グリーンスリーブスを演奏します.

注意： • エラーが生じた場合，音色はDefault VOICE(ハープシコード)に戻ります.

• PLAY文で発生させた音程は，A＝440Hzを基準とした音程と多少誤差があります.

• パラメータを複雑に設定すると音が出なくなります. この場合はSTOPM文を実行してください. それでも音が出ない場合は，プログラムをセーブしたあとリセットスイッチを押してもう一度実行しなおしてください.

POINT

ポイント：point

機能 LP(Last referenced point)を変更する

書式 POINT |(gx , gy) | STEP(sx , sy)|

解説
- N_{80SR}-BASICは，最後にグラフィック操作の行われた座標を覚えています．この基点のことをLP(Last referenced point)といいます．
- POINT 文は，グラフィック操作なしに単にLPを設定する場合に使います．

例 POINT(10 , 30)
- LPの値を(10，30)に設定します．

プログラム例

```
list
100 ' POINT sample
110 SCREEN 0,0:CLS 3
120 FOR I=0 TO 35
130   POINT STEP(8,3)
140   GOSUB 170
150 NEXT I
160 END
170 LINE -STEP(-50,50),1
180 LINE -STEP(100,0),2
190 LINE -STEP(-50,-50),3
200 RETURN
Ok
```

- 画面の左上から右下に向けて三角形を描きます．
- 130行でLPを少しずつ右下に下げていきます．
- 170行から190行で三角形を描きます．

参照：(1) POINT

[1]POINT

ポイント：point

機能 LP(Last referenced point)の値を返す

書式 POINT(〈機能〉)

解説
• 最後にグラフィック操作の行われたグラフィック座標(LP)を，〈機能〉の指定によりグラフィック座標で返します．
• 〈機能〉は0と1の整数値で指定し，返す値は次のようになります．

0：X座標を返します．

1：Y座標を返します．

例
POINT(0) 返される値 ⇒ (X座標)

POINT(1) 返される値 ⇒ (Y座標)

プログラム例

```
list
100 ' POINT sample
110 SCREEN 0,0:CLS 3
120 X=RND(1)*639:Y=RND(1)*199:C=RND(1)*6+1
130 PSET(X,Y),C
140 PRINT "x=";POINT(0);"  y=";POINT(1);
150 PRINT "  color=";POINT(POINT(0),POINT(1))
160 IF INKEY$="" THEN 120 ELSE END
Ok
```

• 何かキーが押されるまで点を描き，その点の座標と色を表示します．
• 120行で，乱数を使って点を描く座標と色を求めて，130行で点を描きます．
• 140行および150行で，POINT関数を使って点を打った座標と色を出力します．

参照：(2) POINT

(2) POINT

ポイント：point

機能 アクティブページ上の指定された座標にあるドットのパレット番号を返す

書式 POINT(gx, gy)

解説 • アクティブページにおいて，グラフィック座標(gx, gy)で指定された座標上にあるドットの色を返します.

• グラフィック画面がカラーモードのときはパレット番号が返され，白黒モードのときは，点があるとき1，ないとき0が返されます.

• 指定された座標がビューポートの外にある場合は−1を返します.

例 POINT(100, 100) 返される値 (グラフィック座標(100, 100)にあるドットのパレット番号：グラフィック画面がカラーモードのとき)

プログラム例

```
list
100 ' POINT sample
110 SCREEN 0,0:CLS 3
120 X=RND(1)*639:Y=RND(1)*199:C=RND(1)*6+1
130 PSET(X,Y),C
140 PRINT "x=";POINT(0);"  y=";POINT(1);
150 PRINT "  color=";POINT(POINT(0),POINT(1))
160 IF INKEY$="" THEN 120 ELSE END
Ok
```

• 何かキーが押されるまで点を描き，その点の座標と色を表示します.

• このプログラム例は(1) POINT関数と同じものです.

POKE

ポーク：poke

機能 メモリ上の指定された番地へデータを書き込む

書式 POKE〈番地〉,〈式〉

解説
- 指定されたメモリ上の番地に1バイトのデータを書き込みます.
- 〈番地〉はデータが書き込まれる番地で，2バイトの整数値(0～65535)の値をとります.
- 〈式〉は書き込まれるデータで，1バイトの整数値(0～255)の値をとります.
- 〈番地〉,〈式〉ともに小数点以下は切り捨てられます.

例 POKE &HD000,&H41
- D000H番地に41Hというデータを書き込みます.

プログラム例

```
list
100 ' POKE sample
110 CLEAR 300,&HDFFF
120 ' write to memory
130 FOR AD=&HE000 TO &HE00F
140   POKE AD,DA:DA=DA+1
150 NEXT AD
160 ' read from memory
170 FOR AD=&HE000 TO &HE00F
180   PRINT RIGHT$(" 0"+HEX$(PEEK(AD)),4);
190 NEXT AD
200 END
Ok
run
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Ok
```

- 直接メモリに0から15までの値を書き込み，それを読み出して出力します.
- 130行から150行で，メモリに0から15まで値を書き込んでいます.

注意：• この命令はメモリの内容を書き換えてしまうため，不用意に使用するとBASICが使っている作業領域を壊してしまい，誤動作の原因となることがあります．メモリマップの状態をよく理解した上で使用してください.

参照：CLEAR, PEEK

PORT

ポート：port

機能　RS-232C通信ポートの入力バッファに入力されている文字数を返す

書式　PORT(〈ポート番号〉)

解説　• RS-232C通信ポートの入力バッファに何文字入力されているかを返します．〈ポート番号〉は1を指定します．

• RS-232C通信ポートの入力バッファは127文字分あり，データがそのバッファの容量を超えた場合"Communications Buffer Overflow"エラーとなります．したがって，PORT関数で，バッファがある程度いっぱいになったかどうかを確認してからINPUT%文などでデータの読み出しを行ってください．

例　PORT(1) 返される値 (RS-232C通信ポートの入力バッファに入力されている文字数)

プログラム例

RS-232C

```
list
100 ' PORT sample
110 INIT %1,&HFA,&H37
120 IF PORT(1) THEN A$=INPUT$(PORT(1),%1) ELSE 120
130 PRINT A$;
140 GOTO 120
Ok
```

• RS-232C通信ポートの入力バッファに文字が入っていた場合にその文字を出力します．

• 120行で，バッファに文字が入っていたら，その文字数だけ変数A$に読み込みます．

参照：INIT%

POS

ポジション：position

機能 カーソルの水平位置を返す

書式 POS(〈式〉)

解説
- テキスト画面上の現在のカーソルの水平位置を返します．
- 〈式〉の値は意味を持ちません．通常は0とします．
- 返される値は，0から，そのときの画面の表示桁数までの整数です．

例 POS(0) 返される値 (カーソルの水平位置)

プログラム例

```
list
100 ' POS sample
110 FOR I=32 TO 127
120   IF POS(0)=>32 THEN PRINT
130   PRINT CHR$(I);" ";
140 NEXT I
150 END
Ok
run
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ ¥ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
Ok
```

- キャラクタコードの32から127までの文字を出力します．
- 120行で，32桁を超えたら改行するようにしています．

参照：CSRLIN, LOCATE, WIDTH

PRESET

ピー・リセット：point reset

機能 グラフィック画面上の任意のドットを消去する

書式 PRESET ｜(gx , gy)｜〔,〈パレット番号〉〕
｜STEP(sx , sy)｜

解説 • 指定したグラフィック座標のドットを消去します．通常〈パレット番号〉は省略し，COLOR文によって設定されている〈バックグラウンドカラー〉でドットを描きます．

• 〈パレット番号〉を指定した場合はPSET文と同じ動作をします．

• PRESET文の実行後，LPは(gx, gy)に移動します．

例 PRESET(30 , 40)

• グラフィック座標(30，40)のドットを消去します．

プログラム例

```
list
100 ' PRESET sample
110 SCREEN 2,0:COLOR ,1:CLS 3
120 LINE(0,100)-(319,100),6
130 LINE(50,0)-(50,200),6
140 'circle
150 P=3.14159
160 FOR T=0 TO 2*P STEP 2*P/100
170   X=50+50*COS(T):Y=100-50*SIN(T)
180   PRESET(X,Y),0
190   X=50+40*T:PRESET(X,Y),0
200 NEXT T
210 END
Ok
```

• 点で，円と正弦曲線を描きます．

• 110行でグラフィック画面を青にします．

• 120行と130行で座標軸を描きます．原点は(50，100)になります．

• 150行は変数Pに円周率(π)を代入しています．

• 160行から200行で円と正弦曲線を描きます．

• ここで描く円は中心座標(50，100)，半径50の円ですから，角度Tとすると，円周上の点(X, Y)は，

$$X=50+50\cos T$$

$$Y=100+50\sin T$$

で得られます．ただし，グラフィック座標のy座標は普通の座標と逆で下にいくほど値が大きくなるので，170行の計算は

$$Y=100-50\sin T$$

で求めています.

参照：COLOR, PSET

PRINT/LPRINT

プリント/エル・プリント：print/lpt print

機　能　数値や文字列を画面やプリンタに出力する

書　式　PRINT〔〈式〉...〕

LPRINT〔〈式〉...〕

解　説

• 指定した〈式〉の値をディスプレイ画面に表示(PRINT)したり，プリンタに出力(LPRINT)したりします．

• 〈式〉が数値式の場合は数値を，文字式の場合は文字列を出力します．〈式〉が省略されていると改行のみを行います(キャリッジリターンとラインフィードを出力します)．

• 〈式〉の区切り記号としてセミコロン(；)を使うと，直前にプリントしたもののすぐ後に続いて次の数値や文字列を出力します。

• PRINT文で値を出力する領域は，あらかじめ各行が14文字毎に分割されています．〈式〉の区切り記号としてコンマ(，)を使うと，この領域毎に出力が行われます．

• 〈式〉の最後にセミコロンおよびコンマを付けると改行動作を起こしません．

• 変数とダブルクォーテーション(")で囲まれた文字列との区切りに限ってセミコロンを省略できます．

• 数値を出力した場合，その後ろには1文字の空白が挿入されます．また，数値の前には符号のための桁を確保します(正の数のとき空白，負の数のとき"－"となります)．

• 単精度の数値で，指数形式でなくても6桁以下の桁数で精度に影響を及ぼさず表示できるものは実数形式で表示されます．同様に倍精度の数値は，16桁以下の桁数で精度に影響を及ぼさず表示できるものは実数形式の表示になります．

• 表示する数値の長さ，文字列の長さが現在のカーソル位置より後方にとれない(それを表示すると次の行にわたってしまう)場合，改行してそれを表示します．

• "PRINT" というキーワードは，入力時に "?" で代用できます．

例　PRINT "computer"

• "computer" という文字列をディスプレイ画面に表示します．

プログラム例

```
list
100 ' PRINT/LPRINT sample
110 INPUT "ﾅﾏｴ ";NA$
120 INPUT "ｾｲﾍﾞﾂ ";SE$
130 INPUT "ﾈﾝﾚｲ ";OD
```

```
140 PRINT "ﾅﾏｴ    :",NA$
150 PRINT "ｾｲﾍﾞﾂ :",SE$
160 PRINT "ﾈﾝﾚｲ   :",OD
170 END
Ok
run ↵
ﾅﾏｴ ? ﾗｲｵﾝ ↵
ｾｲﾍﾞﾂ ? ｵﾄｺ ↵
ﾈﾝﾚｲ ? 2 ↵
ﾅﾏｴ    :          ﾗｲｵﾝ
ｾｲﾍﾞﾂ :          ｵﾄｺ
ﾈﾝﾚｲ   :           2
Ok
```

• 名前，性別，年齢を読み込んで，それを出力します.

• このプログラム例はINPUT文と同じものです.

PRINT

〈ファイル番号〉が1～15の場合

ディスク

プリント・シャープ：print #

機 能 シーケンシャルファイルにデータを出力する

書 式 PRINT #〈ファイル番号〉,〔〈式〉...〕

解 説 •〈ファイル番号〉は，そのシーケンシャルファイルをオープンしたとき指定した番号です．

•カセットテープへ出力するときは〈ファイル番号〉に－1を指定します．なお，この場合，ファイルをオープンする必要はありません．

•〈式〉は，ファイルに書き込まれる数値式や文字式です．〈式〉が2つ以上あるときは，その間をセミコロン(；)またはコンマ(，)で区切ります．

•〈式〉のリストの中の数値式はセミコロンで区切ります．(もし区切り記号にコンマを使うと，数値の間に挿入される余分な空白も書き込まれてしまいます．)

•数値式と文字式を並べて出力する場合，及び文字式の値(文字列)を連続して出力するときは，その間に区切り記号としてコンマを出力させます．そうしないと，文字列を続いて出力したとき，それらのデータは連続した文字列となります．また，文字列自身がデータとしてコンマ，セミコロン，意味のある空白，キャリッジリターン(CHR$(13))，ラインフィード(CHR$(10))などを含む場合は，引用符(CHR$(34))によって囲んで出力します．

例 PRINT #1,CHR$(34)+"width80,25"+CHR$(34);",";"page 3";",";

•"width 80, 25" と "page 3" という2つの文字列をシーケンシャルファイルに書き出す例です．

•"width 80, 25"という文字列は，コンマを含んでいますので引用符(CHR$(34))で囲みます．また，文字列の直後にはコンマを出力し，各値を区切っています．

プログラム例

ディスク

```
list
100 ' PRINT# sample
110 OPEN "2:animal" FOR OUTPUT AS #1
120 INPUT A$
130 IF A$="end" THEN END
140 PRINT #1,A$
150 GOTO 120
160 END
Ok
run
? ﾜﾝﾜﾝ
? ﾆｬｰｵ
? ﾋﾋｰﾝ
? ﾋﾟｰﾋﾟｰ
? end
Ok
```

• ドライブ2に入っているフロッピィディスクに "animal" というシーケンシャルファイルを作ります.

• 140行で，読み込んだ文字列をファイルに書き込んでいます.

• INPUT #文のプログラム例で，ここで作成したファイルの内容を見ることができます.

参照：PRINT# USING, OPEN, INPUT#

PRINT%

プリント・パーセント：print %

機　能　RS-232C通信ポートにデータを出力する

書　式　PRINT %〈ポート番号〉,〔〈式〉...〕

解　説

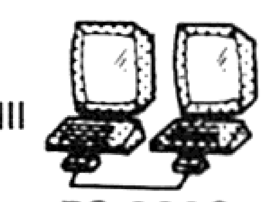
• RS-232C通信ポートにデータを出力します．使い方はPRINT #文と同じです．

•〈ポート番号〉は1を指定します．

例　PRINT %1 , "Hello" ;

• RS-232C通信ポートに "Hello" という文字列を出力します．

プログラム例


RS-232C

```
list
100 ' PRINT% sample
110 INIT %1,&HFA,&H37
120 LINE INPUT "message?";A$
130 PRINT %1,A$
140 GOTO 120
Ok
```

• キーボードから入力したメッセージをRS-232C通信ポートに出力します．

• これは送信側のプログラムですから，受信側はINPUT %文，またはLINE INPUT %文のプログラム例を参照してください．

参照：INIT%

PRINT USING/LPRINT USING

プリント・ユージング/エル・プリント・ユージング：print using/lpt print using

機能 文字列，数値を指定した書式で出力する

書式

```
|PRINT  |USING〈書式制御文字列〉；〈式〉〔|;|〈式〉...〕〔|;|〕
|LPRINT |                                |,|               |,|
```

解説

•〈書式制御文字列〉によって，〈式〉の出力される領域や書式を決定します．

•〈書式制御文字列〉で指定した書式制御の数が〈式〉の数より少ないときは，〈書式制御文字列〉が繰り返し使われます．

文字の書式制御

！…与えられた文字変数や文字定数の最初の１文字だけを出力します．

&＜ｎ個の□(空白)＞&…与えられた文字変数や文字定数の先頭から(n＋2)文字の文字列を出力します．与えられた文字列が(n＋2)文字より長い場合は余分な文字は無視され，短い場合には文字列は左詰めに出力され，残った部分には空白が出力されます．

数値の書式制御

#…数値を出力する桁数を指定します．指定した桁数より数値の桁数が小さいときには右詰めで出力されます．

．…小数点の位置を指定します．小数点以下の部分で有効ではない桁には０が出力されます．

＋…〈書式制御文字列〉の最初または最後に付けた場合，数値の符号がそれぞれ前または後に出力されます．２個以上の"＋"を並べた場合には，余分の"＋"は後述の制御文字以外の文字と同じ扱いになります．

－…〈書式制御文字列〉の最後に付けた場合，数値が負の数のときに数値の後ろに"－"が出力されます．前に付けたり，２個以上並べた場合には，後述の制御文字以外の文字と同じ扱いになります．

＊＊…〈書式制御文字列〉の先頭に付けた場合，数値領域の左側に空白部分ができたとき，そこを"＊"で埋めて出力します．この"＊＊"は２桁分の領域を確保します．

￥￥…〈書式制御文字列〉の先頭に付けた場合，出力される数値の直前に"￥"を出力します．"￥￥"は２桁分の領域を確保しますが，このうち１桁分は"￥"の出力領域として使われます．後述の指数形式の書式指定を行った場合には，正しく出力されないことがあります．

＊＊¥…〈書式制御文字列〉の先頭に付けた場合，上記の2つ(＊＊と¥¥)の両方の機能となります．"＊＊¥"は3桁分の領域を確保しますが，このうち1桁分は"¥"の出力領域として使われます．

,… 桁数指定の"#"の並びの中に置いた場合，数値の整数部が3桁毎に","で区切られて出力されます．ただし，"."より右側に置いた場合は，数値の最後に","が出力され，3桁毎の区切りは行われません．

^^^^…桁数指定の"#"の後に付けた場合，数値は指数形式で出力されます．

制御文字以外の文字

以上の制御文字以外の文字(英数字，カナ，グラフィック記号等)を置いた場合，数値の前や後ろにそのキャラクタが出力されます．

数値の領域を超えた場合

指定した数値領域より数値の桁数が大きい場合，数値の直前に"%"が出力されます．また，指定した数値領域より数値の小数点以下の桁数が大きい場合，指定した小数点領域の次の桁数で四捨五入を行います．たとえば領域を"###.#"としたとき数値に123.45が指定されていた場合，小数点第2位で四捨五入され，123.5になります．したがって，四捨五入された数値が領域より大きくなる原因となった場合も，数値の前に"%"が出力されます．

例

PRINT USING "###.##";A,B

• 数値変数A，Bの値を5桁(小数点以下2桁)で出力します．

プログラム例

```
list↵
100 ' PRINT USING sample
110 PRINT USING "!";"NEC Computer"
120 PRINT USING "&              &";"NEC Computer"
130 PRINT USING "#####";123.456
140 PRINT USING "#####.#";123.456
150 PRINT USING "+####.#";123.456
160 PRINT USING "#####.#-";-123.456
170 PRINT USING "**###.#";123.456
180 PRINT USING "¥¥###.#";123.456
190 PRINT USING "**¥##.#";123.456
200 PRINT USING "#####,.#";1234.6
210 PRINT USING "#####";123456!
220 END
Ok
run↵
N
NEC Computer
  123
  123.5
 +123.5
  123.5-
**123.5
 ¥123.5
```

```
*¥123.5
 1,234.6
%123456
Ok
```

• 指定した文字列あるいは数値を指定した領域や書式で出力します.

PRINT# USING

〈ファイル番号〉が1～15の場合

プリント・シャープ・ユージング：print # using

機能 文字列，数値を指定した書式でファイルに出力する

書式 PRINT #〈ファイル番号〉, USING〈書式制御文字列〉;〈式〉〔{;|,}〈式〉...〕〔{;|,}〕

解説
- 〈ファイル番号〉で指定されたファイルに対して，文字列や数値を書式指定して出力します．
- PRINT # USING文は，その対象がファイルであることを除けば PRINT USING文と同じです．
- カセットテープへ出力する場合は〈ファイル番号〉に－1を指定します．なお，この場合はファイルをオープンする必要はありません．

例 PRINT #2, USING "####"; A

- 数値変数Aの値を4桁右詰めでファイルに書き込みます．

プログラム例

```
list
100 ' PRINT# USING sample
110 OPEN "stest.dat" FOR OUTPUT AS #1
120 FOR I=0 TO 10
130   PRINT #1,USING "###";I^2
140 NEXT I
150 END
Ok
```

- 0から10までの2乗の値を3桁ずつ右詰めでファイルに書き込みます．

参照：PRINT USING, PRINT #, OPEN

PSET

ピー・セット：point set

機能　グラフィック画面上のドットを描く

書式

```
PSET |(gx,gy)       |〔,〈パレット番号〉〕
     | STEP(sx,sy)  |
```

解説

• 指定したグラフィック座標にドットを描きます.

• ドットの色は〈パレット番号〉によって指定し，省略された場合にはCOLOR文の〈フォアグラウンドカラー〉が用いられます.

• PSET文の実行後，LPは(gx, gy)に移動します.

例　PSET(30,40),5

• グラフィック座標(30, 40)にパレット番号5の色でドットを描きます(カラーモードのとき).

プログラム例

```
list↵
100 ' PSET sample
110 SCREEN 2,0:COLOR ,0,,5:CLS 3
120 LINE(0,100)-(319,100),6
130 LINE(50,0)-(50,200),6
140 'circle
150 P=3.14159
160 FOR T=0 TO 2*P STEP 2*P/100
170   X=50+50*COS(T):Y=100-50*SIN(T)
180   PSET(X,Y)
190   X=50+40*T:PSET(X,Y)
200 NEXT T
210 END
Ok
```

• 点で円と正弦曲線を描きます.

• 詳しい説明は，PRESET文のプログラム例を参照してください.

参照：COLOR, PRESET

PUT#

ディスク

プット・シャープ：put #

機　能　バッファ中のデータをランダムファイルに書き出す

書　式　PUT〔#〕〈ファイル番号〉〔,〈数式〉〕

解　説　•〈ファイル番号〉で指定されたバッファの内容をランダムファイルへ書き込みます．指定されたファイルは，ランダムファイルのモードでオープンされていなければなりません．

•〈数式〉はファイルのレコード番号です．〈数式〉が省略された場合，直前に行われたGET #文，PUT #文で参照されたレコードの次に書き込まれます．なお，書き出すデータはあらかじめFIELD文，LSET/RSET文で準備しておかなければなりません．

例　PUT #3,5

•ファイル番号3でオープンしたランダムファイルの5番目のレコードに，現在バッファに入っているデータを書き込みます．

プログラム例

ディスク

```
list ↵
100 ' PUT# sample
110 OPEN "2:address" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 LINE INPUT "NAME? ";NA$
140 IF NA$="end" THEN 200
150 LINE INPUT "TEL? ";TL$
160 LINE INPUT "ADDRESS? ";AR$
170 LSET A$=NA$:LSET B$=TL$:LSET C$=AR$
180 PUT #1
190 GOTO 130
200 END
Ok
run ↵
NAME? ｱｲｶﾜ ﾌｻｺ ↵
TEL? 115-787-9898 ↵
ADDRESS? ｶﾅｶﾞﾜｹﾝ ｶﾜｻｷｼ ﾄｶｲ ↵
NAME? ﾂﾁﾔ ﾋﾃﾞﾐ ↵
TEL? 114-123-6565 ↵
ADDRESS? ﾄｳｷｮｳﾄ ｼﾅｶﾞﾜｸ ｲﾅｶ ↵
NAME? end ↵
Ok
```

•住所録のファイルを作ります．

•170行で，入力したデータをバッファにセットして，180行でフロッピィディスクに書き込んでいます．

参照：OPEN, FIELD, LSET/RSET, GET

PUT@

プット・アットマーク：put @

機能　グラフィック画面にグラフィックパターンや漢字を表示する

書式　(1) PUT@ ｜(gx , gy)｜ ,〈配列変数名〉〔(〈要素ナンバ〉)〕〔,〈機能〉〕
｜STEP(sx , sy)｜
〔,〈フォアグラウンドカラー〉〕〔,〈バックグラウンドカラー〉〕

(2) PUT@ ｜(gx , gy)｜ , KANJI(〈漢字コード〉)〔,〈機能〉〕〔,〈フォア
｜STEP(sx , sy)｜
グラウンドカラー〉〕〔,〈バックグラウンドカラー〉〕

解説　•(1)の書式は，GET@文によって配列に読み込まれたグラフィックパターンをグラフィック画面上の任意の位置に表示します．

•〈配列変数名〉は，表示したいグラフィックパターンが格納されている配列名であり，〈要素ナンバ〉は配列内のどこからデータを取り始めるかの指定です．〈要素ナンバ〉が省略された場合は，配列の最初(〈要素ナンバ〉は0)から取り始めます．これらの指定は，GET@文で使ったものを指定します．

•〈機能〉とは，グラフィックパターンを画面に表示する際，いろいろな機能を指定するもので，次に示すものが用意されています．

PSET	配列内のグラフィックパターンをそのまま表示します．
PRESET	白黒モードの場合は配列内のパターンを反転させて表示します． カラーモードの場合は，各ドットのパレット番号を7−(そのドットのパレット番号)として表示します．
OR	配列内のグラフィックパターンと，既に画面上に表示されているグラフィックパターンを1ドットごとにOR(論理和)し，その結果を画面に表示します．
AND	配列内のグラフィックパターンと画面上のパターンをドットごとにAND(論理積)し，その結果を画面上に表示します．
XOR	配列内のパターンをドットごとにXOR(排他的論理和)し，その結果を画面に表示します．

•これらの〈機能〉は画面モードによって演算の対象が違います．白黒モードの場合，ドットがあるかないかを対象とし，カラーモードの場合，パレット番号を対象とします．もし〈機能〉が省略された場合はXORとみなされます．

•〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉は，白黒モードのときに読み込んだパターンをカラーモードにおいて表示するときにのみ有効なオプションパラメータです．

•〈フォアグラウンドカラー〉は，白黒モードで読み込んだ際に白であったドットに対しての色指定で，これをカラーモードにおいて実行すると任意の色に変えることができます．省略された場合，COLOR文の〈フォアグラウンドカラー〉が用いられます．

•〈バックグラウンドカラー〉は，同様に黒であったドットに対しての色指定です．省略された場合，COLOR文の〈バックグラウンドカラー〉が用いられます．これらはともに0から7のパレット番号によって指定します．

•GET@文とPUT@文は，上記した〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉を指定して白黒モードで読み込んだパターンをカラーモードで表示する用途の他は，原則として同一画面モードで使用するようにしてください．

•PUT@文の実行後，LPは(gx，gy)に移動します．

•(2)の書式は，別売の漢字ROMボードを接続することによって，〈漢字コード〉で指定された漢字などの文字を画面に表示します．

•この書式では，配列内のデータに当たるものをあらかじめ用意することを除いて，その使い方および機能は(1)の書式の場合とほぼ同様です．ただし，漢字パターンは原則として縦16ドット，横16ドットの大きさと決められています(8×8，8×16のものもあります)．また，漢字パターンは白黒モードで読み込んだ場合と同じ形で格納されていますから，カラーモードにおいて〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉で指定してカラー表示させることも可能です．

例　PUT@(10,10),G%,PSET

•グラフィック座標(10，10)を始点として，配列G%に格納されているグラフィックパターンを表示します．

プログラム例

次のプログラムを実行する場合は，別売の漢字ROMボードが必要です．

```
list↵
100 ' PUT@ sample
110 SCREEN 2:COLOR 2
120 AD=&H3021:CLS 3
130 LOCATE 0,0:PRINT "JIS CODE =";HEX$(AD);"H"
140 FOR Y=22 TO 122 STEP 20
150   FOR X= 2 TO 302 STEP 20
160     PUT@(X,Y),KANJI(AD),PSET
170     AD=AD+1
```

```
180      IF AD MOD 256=&H7F THEN AD=AD+&HA2
190    NEXT X
200 NEXT Y
210 IF AD>&H4F5F THEN END ELSE 130
Ok
```

• JISコードの第1水準の漢字を表示します.

• 160行で，変数ADの値に対応する漢字を出力します.

参照：GET@

READ

リード：read

機能 DATA文中に格納されている値を変数に読み込む

書式 READ〈変数名リスト〉

解説 • READ文は，DATA文中に格納されている定数を変数に読み込みます．READ文の変数は数値変数でも文字変数でもかまいませんが，対応するDATA文中の定数の型と一致していなければなりません．

• プログラムの実行中読み出される定数の個数が，DATA文で定義されているものより多い場合，"Out of DATA" エラーとなります．

• RESTORE文を実行することにより，読み出すDATA文の位置を行単位で指定できます．

例 READ NM$，AG

• DATA文中に格納されている文字定数，数値定数を変数NM$，AGに読み込みます．

プログラム例

```
list
100 ' READ sample
110 FOR I=0 TO 3
120   READ A:PRINT A;
130 NEXT I
140 READ A$:PRINT A$
150 END
160 DATA 3,2,1,0,Fire !
Ok
run
 3  2  1  0 Fire !
Ok
```

• " 3 2 1 0 Fire !" という文字列を出力するプログラムです．

• 110行から130行でDATA文から数値データを読み込んで出力し，140行でDATA文から "Fire !" という文字列を読み込んで出力します．

参照：DATA，RESTORE

REM

リマーク：remark

機能 プログラム中に注釈を入れる

書式 REM〔〈注釈文〉〕

'〔〈注釈文〉〕

解説

• REM文は，プログラムを見やすくしたり，コメントを書いたりするために使われます．プログラムの実行には影響を与えません．

• REM文ではコロン(：)も注釈の一部となりますので，コロンで区切って他の文を続けることはできません．

• REM文は，BASICによって実行される文の後にコロンで区切って付け加えることができます．

例 REM ＊＊＊ COMMENT ＊＊＊

• 注釈行を示します．

プログラム例

```
list↵
100 ' REM sample
110 REM    REM ｽﾃｰﾄﾒﾝﾄ ﾆ ﾖｯﾃ ﾌﾟﾛｸﾞﾗﾑﾉ ﾅｶﾆ
120 REM    ｺﾒﾝﾄ ｶﾞ ｶｹﾏｽ｡
130 '      REM ﾊ "'" ﾃﾞ ﾀﾞｲﾖｳ ﾃﾞｷﾏｽ｡
140 END
Ok
run↵
Ok
```

• このプログラムはすべて注釈文のため何も実行しません．

• 100行と130行では "'" でREM文の代用をしています．

RENUM

リナンバー：renumber

機能 プログラムの行番号を付け替える

書式 RENUM〔〈新行番号〉〕〔,〈旧番号〉〕〔,〈増分〉〕

解説

•〈新行番号〉は，新しく付ける行番号の最初の行番号で，省略時は10とみなされます．

•〈旧行番号〉は，行番号の付け替えを始める現在のプログラムの行番号です．省略したときは，そのプログラムの最初の行番号です．

•〈増分〉は，新しく付ける各行番号の間の増分で，省略したときは10とみなされます．

• RENUMコマンドは，GOSUB文，GOTO文，ON . . . GOSUB文，ON . . . GOTO文およびERL変数などで参照している行番号も新しい行番号に対応して変更します．これらの文が参照している行番号の行がRENUMコマンドを実行する前のプログラム中に存在しない場合には "Undefined line xxxxx in yyyyy" というメッセージが表示されます．
この場合，存在しなかった行番号(xxxxx)はRENUMコマンドによって変更されませんが，その文の行番号(yyyyy)は変更されます．

例 RENUM 1000

• メモリ上にあるプログラムの行番号を付け直します．実行後，行番号は1000, 1010, 1020, . . . となります．

実 行 例

```
list⏎
100 ' RENUM sample
110 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲｼﾀ ｷﾞｮｳ ｶﾗ
120 ' ｷﾞｮｳ ﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴ ﾏｽ｡
130 '
140 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲｼﾀ ｷﾞｮｳ ｶﾗ"
150 PRINT " ｷﾞｮｳ ﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴ ﾏｽ｡"
160 END
Ok
renum 200,130,20⏎
Ok
list⏎
100 ' RENUM sample
110 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲｼﾀ ｷﾞｮｳ ｶﾗ
120 ' ｷﾞｮｳ ﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴ ﾏｽ｡
200 '
220 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲｼﾀ ｷﾞｮｳ ｶﾗ"
240 PRINT " ｷﾞｮｳ ﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴ ﾏｽ｡"
260 END
Ok
```

• 130行以降の行番号を，200行から20行ごとに付け替えます．

RESTORE

リストア：restore

機能 READ文で読むDATA文の位置を指定する

書式 RESTORE〔〈行番号〉〕

解説
• READ文で読み込みを開始するDATA文の位置を行単位で指定します．
• 〈行番号〉を指定すると，指定された行以降のDATA文の内容から読み始めます．〈行番号〉を省略したときは，プログラム中の最初のDATA文の内容から読み始めます．

例 RESTORE 800
• READ文で読み出されるDATA文の位置を800行に設定します．

プログラム例

```
list
100 ' RESTORE sample
110 READ A,B
120 RESTORE:READ C,D
130 RESTORE 180:READ E$,F$
140 PRINT A,B:PRINT C,D:PRINT E$,F$
150 END
160 DATA 1,2
170 DATA 3,4
180 DATA AA,BB
Ok
run
 1             2
 1             2
AA            BB
Ok
```

• 160行のデータを2回出力し，次に180行のデータを出力するプログラムです．
• 120行のRESTORE文によって，次のREAD文は160行のデータから読むことになります．
• 130行では，170行のDATA文を飛ばして180行のデータを読み込んでいます．

参照：DATA, READ

RESUME

リジュウム：resume

機能 エラー回復処理終了後，プログラムの実行を再開する

書式 RESUME

RESUME NEXT

RESUME〈行番号〉

解説 • RESUME文は，ON ERROR GOTO文によってエラー処理ルーチンに分岐したあと，プログラムの実行を再開する場合に用います．

• プログラムの実行を再開する場所に応じて次のように3つの書式を選ぶことができます．

RESUME……………………エラーの原因となった文からプログラムの実行が再開されます．このときエラーとなった原因を取り除かないと，再びエラー処理ルーチンへ戻ります．

RESUME NEXT………エラーの原因となった文のすぐ次の文から実行が再開されます．

RESUME〈行番号〉…〈行番号〉で指定した行から実行が再開されます．

例 RESUME 1000

• エラー処理ルーチンでエラー回復処理後，1000行から実行を再開します．

プログラム例

```
list
100 ' RESUME sample
110 ON ERROR GOTO 300
120 INPUT "X,Y";X,Y
130 A=X/Y
140 PRINT "X/Y =";A
150 END
300 PRINT "** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **"
310 RESUME 120
Ok
run
X,Y? 100,10
X/Y = 10
Ok
run
X,Y? 35,0
** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **
X,Y? 35,5
X/Y = 7
Ok
```

• 2つの実数X，Yを読み込んでX/Yを出力します．

• 130行の除算でエラーが生じた場合，300行でメッセージを出力して，310行の

RESUME文で120行から実行を再開します.

参照：ON ERROR GOTO

RETURN

リターン：return

機能 サブルーチンから抜ける

書式 RETURN

解説 • GOSUB文によって呼ばれたサブルーチンから抜け出し，GOSUB文の次の文へ実行を移します．

• RETURN文はGOSUB文と正しく対応していなければなりません．もし単独で実行した場合，"RETURN without GOSUB" エラーとなります．

例 RETURN

• サブルーチンから抜け出します．

プログラム例

```
list
100 ' RETURN sample
110 INPUT "ﾃｲﾍﾝ  :";BA
120 INPUT "ﾀｶｻ   :";H
130 GOSUB 160
140 PRINT "ﾒﾝｾｷ ﾊ";AR
150 END
160 AR=BA*H/2
170 RETURN
Ok
run
ﾃｲﾍﾝ  :? 78
ﾀｶｻ   :? 8
ﾒﾝｾｷ ﾊ 312
Ok
```

• 三角形の底辺と高さを読み込んで面積を出力するプログラムです．

• このプログラム例はGOSUB文と同じものです．

参照：GOSUB

RIGHT$

ライト・ダラー：right $

機　能　文字列の右側から指定された長さの文字列を返す

書　式　RIGHT$(〈文字列〉,〈数式〉)

解　説
- RIGHT$関数は〈文字列〉の右側から〈数式〉で指定された長さの文字列を取り出し，それを値として返します．
- 〈数式〉の範囲は0～255です．
- 〈数式〉が〈文字列〉の総文字数以上の場合は〈文字列〉全体を，また〈数式〉が0ならばヌルストリングを返します．

例　RIGHT$(TIME$,2) 返される値 (TIME$変数の右側2文字)
- カレンダ時計の“秒”だけを取り出します．

プログラム例

```
list
100 ' RIGHT$ sample
110 INPUT A$
120 FOR I=1 TO 10
130   PRINT RIGHT$(A$,I)
140 NEXT I
150 END
Ok
run
? 0123456789
9
89
789
6789
56789
456789
3456789
23456789
123456789
0123456789
Ok
```

- 読み込んだ文字列を，右から1文字，2文字……10文字表示します．
- 130行で，変数A$の右からI番目までを出力します．

参照：LEFT$, MID$

RND

ランド：random

機　能　乱数の値を返す

書　式　RND(〈数式〉)

解　説
- 0以上1未満の乱数値(単精度実数型)を返します.
- 発生される乱数は,〈数式〉の値によって次のようになります.

　〈数式〉が負の場合——あらかじめ決められた乱数系列の初期値に戻します.

　〈数式〉が0の場合——1つ前に発生した乱数の値をとります.

　〈数式〉が正の場合——次の乱数を発生します.

例　RND(1) 返される値⇒ (乱数の値)

プログラム例

```
list↵
100 ' RND sample
110 DIM SUM(6)
120 FOR I=1 TO 100
130   DA=INT(RND(1)*6+1)
140   SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170   PRINT I"-";SUM(I);"%"
180 NEXT I
190 END
Ok
run↵
 1 - 22 %
 2 - 15 %
 3 - 13 %
 4 - 12 %
 5 - 17 %
 6 - 21 %
Ok
```

- 1から6までの数を乱数で発生させて,それぞれ発生した割合を出力するプログラムです.
- 130行で1から6までの整数を発生させています.

ROLL

ディスク

ロール：roll

機能 グラフィック画面をスクロールさせる

書式 ROLL〈スクロール数〉〔,〈方向〉〕

解説

• グラフィック画面を上下左右方向にスクロールさせます．

•〈スクロール数〉で，スクロールさせるドット数(上下方向)，バイト数(左右方向)を指定します．左右方向のスクロールは8ドットが1つの単位となりますので，これをバイト数と呼びます．

•〈スクロール数〉の値の範囲は次のとおりです．

上下方向……−200〜200
左右方向……−80〜80(640×200ドットモード)
　　　　　　−40〜40(320×200ドットモード)

なお，負の数を指定した場合は，指定した〈方向〉とは反対方向に絶対値の分だけスクロールします．

•〈方向〉は，スクロールさせる方向を0〜3の値で指定します．

```
        0(上)
          ↑
(左)1←  ○  →3(右)
          ↓
        2(下)
```

省略されたときは0(上)とみなされます．

• スクロールによって表示されなくなったグラフィックデータは消去され，新たに実画面に入った領域はバックグラウンドカラーで塗りつぶされます．

例 ROLL 16

• グラフィック画面を上方向へ16ドットスクロールさせます．

プログラム例

ディスク

```
list↵
100 ' ROLL sample
110 SCREEN 0,0:CLS 3:SCREEN 2,,,3
120 C=0
130 FOR Y=0 TO 119 STEP 3
140   FOR I=0 TO 1
150     SCREEN ,,I
160     LINE(0,Y)-(319,Y),(C MOD 7)+1
170   NEXT I
180   C=C+1
```

```
190 NEXT Y
200 FOR K=1 TO 40
210   SCREEN ,,0:ROLL 5,0
220   SCREEN ,,1:ROLL 5,2
230 NEXT K
240 SCREEN 0:END
Ok
```

• カラーモードⅡで2画面に線を描き，一方の画面を上に，もう一方の画面を下にスクロールさせます.

• 210行で0ページを上に5ドット分スクロールさせ，220行で1ページを下に5ドット分スクロールさせます.

RUN

ディスク 書式(2)の場合

ラン：run

機能 プログラムを実行する

書式 (1) RUN〔〈行番号〉〕

(2) RUN〈ファイル名〉〔，R〕

解説 • RUNの後に何も指定しない場合は，メモリ上にあるプログラムの先頭から実行を開始します．

• 〈行番号〉を指定すると，その行から実行が開始されます．

• 〈ファイル名〉を指定すると，フロッピィディスクから指定した〈ファイル名〉のプログラムをロードし，実行します．

• RUNコマンドを実行すると，すべての開いているファイルを閉じますが，Rオプションを付けた場合には，すべてのファイルは開いたままになります．

例 RUN "demo"

• ドライブ1のフロッピィディスク上にある "demo" というファイル名のプログラムをロードし，実行します．

実行例

ディスク

(a) `run↵`

(b) `run 1000↵`

(c) `run"2:test.nsr"↵`

(d) `run"2:test.nsr",r↵`

• (a)は，現在メモリ上にあるプログラムの先頭から実行を始めます．

• (b)は，現在メモリ上にあるプログラムの1000行から実行を始めます．

• (c)は，ドライブ2のフロッピィディスクに入っている "test. nsr" というプログラムをロードして，先頭から実行を始めます．

• (d)は，現在オープンしているファイルを開いたまま，ドライブ2のフロッピィディスクに入っている"test. nsr" というプログラムをロードして，先頭から実行を始めます．

SAVE

ディスク

セーブ：save

機　能　メモリ上のプログラムをフロッピィディスクにセーブする

書　式　SAVE〈ファイル名〉〔,A〕

解　説　•〈ファイル名〉で指定されるファイルにメモリ上のプログラムを書き込みます．指定したファイル名と同じ名前のファイルが存在した場合には，古い内容は失われて新しいものに変更されます．

•Aオプションが指定された場合には，プログラムはアスキー形式でセーブされます．オプションの指定がない場合には，バイナリ形式に圧縮されてプログラムのセーブが行われます．アスキー形式のセーブは，バイナリ形式よりも多くのファイルスペースを必要としますが，MERGEコマンドのように，セーブされたプログラムファイルを操作する場合に用いられます．また，アスキー形式でセーブされたファイルは，データファイルとして読み出すことができます．

例　SAVE "2:sample"

•ドライブ2に入っているフロッピィディスクに，メモリ上にあるプログラムを"sample"というファイル名でセーブします．

S

実　行　例

ディスク

(a)
```
save "2:test4.nsr"↵
Ok
```
(b)
```
save "2:test3.nsr",a↵
Ok
```

•(a)は，現在メモリ上にあるプログラムを"test4.nsr"というファイル名でドライブ2に入っているフロッピィディスクにセーブします．

•(b)は，現在メモリ上にあるプログラムを"test3.nsr"というファイル名でドライブ2に入っているフロッピィディスクにアスキー形式でセーブします．

参照：LOAD, MERGE

SCREEN

スクリーン：screen

機能 グラフィック画面に対して種々のモード設定を行う

書式 SCREEN〔〈画面モード〉〕〔,〈画面スイッチ〉〕〔,〈アクティブページ〉〕〔,〈ディスプレイページ〉〕

解説 •〈画面モード〉は，グラフィック画面の最も基本的なモードを設定するものです．PC-8001MKⅡSRは4種類の画面モードを持ち，それぞれ次のように設定します．

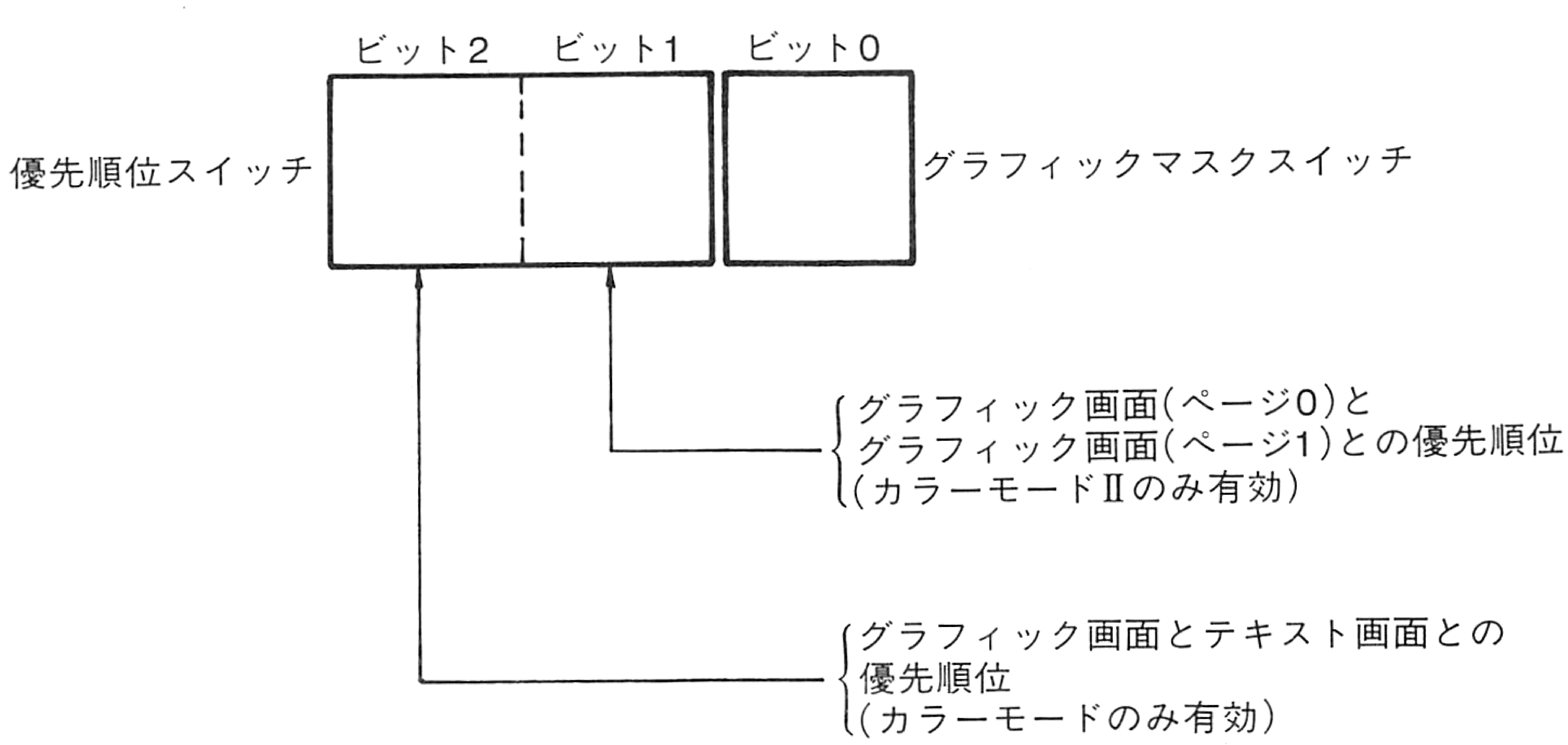
0：カラーモードⅠ(640×200ドット，1画面)
1：白黒モードⅠ　(640×200ドット，3画面)
2：カラーモードⅡ(320×200ドット，2画面)
3：白黒モードⅡ　(320×200ドット，6画面)

•〈画面スイッチ〉は，1ビットで表現される"グラフィックマスクスイッチ"と2ビットで表現される"優先順位スイッチ"の2つのスイッチを合わせ持ったパラメータです．

•"グラフィックマスクスイッチ"を1にすると，グラフィック画面の表示を禁止します．ただし表示内容は残っていますので，このスイッチを0にすることによって再表示することができます．

•"優先順位スイッチ"は，カラーモードにおいて，グラフィック画面とテキスト画面間に表示の際の優先順位を決めるものです．

•〈画面スイッチ〉の2つのスイッチは次のように3ビットに対応しており，0～7の値で指定します．



パラメータの値		画面の優先順位	グラフィック画面
10進	2進		
0	000	TX＞G0＞G1	表示する
1	001		表示しない
2	010	TX＞G1＞G0	表示する
3	011		表示しない
4	100	G0＞G1＞TX	表示する
5	101		表示しない
6	110	G1＞G0＞TX	表示する
7	111		表示しない

TX: テキスト画面
G0: グラフィック画面(ページ0)
G1: グラフィック画面(ページ1)

•〈アクティブページ〉は，グラフィック命令で操作の対象となるページのことです．指定する値はグラフィック画面のモードによって異なり，次のようになります．

カラーモードⅠ：0(常に0を指定します)
白黒モードⅠ　：0〜2
カラーモードⅡ：0〜1
白黒モードⅡ　：0〜5

どのモードにおいても上記の範囲を超えて指定した場合，"Illegal function call" エラーとなります．

•〈ディスプレイページ〉は，複数のページのうちどのページを表示するかを指定します．この指定は各ページがビットに対応した6ビットの2進数表現(0〜63)で行います．

ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
ページ5	ページ4	ページ3	ページ2	ページ1	ページ0

たとえば，

　　0＝000000(2)のとき→全ページ表示しない

　　4＝000100(2)のとき→ページ2のみ表示する

　　5＝000101(2)のとき→ページ0とページ2を合成して表示します

　63＝111111(2)のとき→全ページ表示します

となります．

このパラメータの値の範囲はグラフィック画面のモードによって異なり，次のようになります．

　　カラーモードⅠ：0〜1

　　白黒モードⅠ　：0〜7

　　カラーモードⅡ：0〜3

　　白黒モードⅡ　：0〜63

どのモードにおいても上記の範囲を超えて指定した場合，"Illegal function call" エラーになります．

• この〈アクティブページ〉と〈ディスプレイページ〉の設定は気を付けないと，間違ってページを設定したり，書き込んだはずなのに表示ページになっていないなどのため，思ったとおりに表示できないことがあります．

• SCREEN文によって画面モードを設定すると，設定されているビューポート，LPの値は初期状態に戻されます．つまり，ビューポートは，

　　(0,0)−(639,199)･･･640×200ドットモード

　　(0,0)−(319,199)･･･320×200ドットモード

に，またLPの値は(0, 0)に設定されます．

• 〈画面モード〉を，640×200ドットモード(カラーモードⅠ，白黒モードⅠ)から320×200ドットモード(カラーモードⅡ，白黒モードⅡ)へ変更した場合，またはその逆の場合，グラフィック画面が消去されます．

• 白黒モードにおいて，グラフィック命令で〈パレット番号〉を指定した場合は，0か0以外かによって黒か白かを判定します．また〈パレット番号〉を省略した場合は，カラーモードと同様，COLOR文によって設定されている〈フォアグラウンドカラー〉のパレット番号が採用されます．その際も，0か0以外かの判定により黒，白を決定します．

• システム起動時のモード設定は，

　　SCREEN 0,0,0,1（カラーモードⅠ，グラフィック画面表示，テキスト画面優先，アクティブページ0，ディスプレイページ全画面）

となっています．

例　SCREEN 2,4

• グラフィック画面のモードをカラーモードⅡに設定します．さらに表示の際の優先順位を，

グラフィック画面ページ0＞グラフィック画面ページ1＞テキスト画面

とします．

プログラム例

```
list
100 ' SCREEN sample
110 SCREEN 0,0:CLS 3:SCREEN 3,0,0,63
120 FOR R=0 TO 300 STEP 60
130   X=40*COS(R/180*3.14)+160
140   Y=100-40*SIN(R/180*3.14)
150   SCREEN ,,R/60
160   CIRCLE(X,Y),9,7
170   PAINT(X,Y),1,1
180 NEXT R
190 FOR I=1 TO 10
200   FOR P=0 TO 5
210     SCREEN ,,,2^P
220     FOR W=0 TO 10:NEXT W
230   NEXT P
240 NEXT I
250 END
Ok
```

• 白黒モードⅡで，それぞれの画面に一つずつ円を描き，画面を順番に表示させることによって，円が動いているように見せます．

参照：1.14　画面モード

SET

ディスク

セット：set

機能 ファイルまたはフロッピィディスクの属性を設定する

書式 SET〈ドライブ番号〉,〈"属性文字"〉
SET #〈ファイル番号〉,〈"属性文字"〉
SET〈ファイル名〉,〈"属性文字"〉

解説 • 指定したファイルやフロッピィディスクに，〈"属性文字"〉によって指定された属性を付けます．属性文字は大文字の "P", "R" で指定し，それぞれライトプロテクト(書き込み禁止)，リードアフターライト(書き込み確認)の属性を付けます．これ以外の文字が属性文字として指定された場合には，現在設定されている属性がすべて解除されます．

• 〈ドライブ番号〉が指定された場合には，指定されたドライブの中に入っているフロッピィディスクに対して属性が付けられます．

• 〈ファイル番号〉が指定された場合には，以後そのファイルへの出力に対して指定された属性が作用します．

• 〈ファイル名〉が指定された場合には，指定されたファイルのみに属性が付けられ，同一のフロッピィディスク中の他のファイルにはその影響は及びません．

• P属性を付けると，PRINT #文，PUT #文などの書き込み動作が禁止されるとともに，ファイルの削除もできなくなります．

• R属性を付けると，書き込みを行った直後に読み出しを行い，正しく書き込みが行われたかの確認がされるようになります．確認の結果，正しくなかった場合は "Disk I/O error" エラーになります．

例 SET 2,"P"

• ドライブ2に入っているフロッピィディスクに書き込み禁止の属性を付けます．

実 行 例

ディスク

```
set "2:test.nsr","P"↵
Ok
kill "2:test.nsr"↵
File write protected
Ok
```

• 実行例で，始めにドライブ2のフロッピィディスクに入っている "test.nsr" というファイルにSET文でライトプロテクトをかけているため，KILLコマンドではファイルを消すことができません．

参照：ATTR$

SGN

エス・ジー・エヌ：sign

機　能　数値の符号を返す

書　式　SGN(〈数式〉)

解　説
- 〈数式〉が正の場合は1を，0の場合は0を，負の場合は−1を返します．
- 〈数式〉には，整数型，単精度実数型，倍精度実数型が指定できます．

例

SGN(0.5)　返される値 ⇒ 1

SGN(0)　返される値 ⇒ 0

SGN(−0.5)　返される値 ⇒ −1

プログラム例

```
list⏎
100 ' SGN sample
110 SCREEN 0,0:CLS 3
120 LINE(0,100)-(639,100),1
130 X=5
140 FOR I=0 TO 31.42 STEP .05
150   S=SGN(INT(COS(I)*20))
160   ON S+2 GOSUB 200,220,240
170   X=X+1
180 NEXT I
190 END
200 'plus
210 PSET(X,20),5:RETURN
220 'zero
230 LINE(X,20)-(X,179),5:RETURN
240 'minus
250 PSET(X,179),5:RETURN
Ok
```

- 矩形波を描きます．
- 140行からのFOR文の制御変数Iは，角度(ラジアン)を表します．
- 150行で余弦の値が正か負かを調べ，正なら画面の上方，負なら画面の下方に点を打ちます．

SIN

サイン：sine

機　能　正弦(サイン)の値を返す

書　式　SIN(〈数式〉)

解　説
- 〈数式〉の値の正弦(サイン)を返します.
- 〈数式〉の値の単位はラジアン(π/180×角度)です.
- SIN関数の演算は単精度で行われます.

例

SIN(3.14159/180＊30)　返される値 ⇒　0.5(SIN(30°)の値)

SIN(3.14159/180＊60)　返される値 ⇒　0.866025(SIN(60°)の値)

プログラム例

```
list
100 ' SIN sample
110 F$="CSEC(###°)=###.#####"
120 INPUT "ｶｸﾄﾞ  :";D
130 R=D MOD 180
140 IF R=0 THEN PRINT "ﾃｲｷﾞｻﾚﾏｾﾝ":END
150 C=1/SIN(R/180*3.1416)
160 PRINT USING F$;D,C
170 END
Ok
run
ｶｸﾄﾞ  :? 15
CSEC( 15°)=   3.86370
Ok
run
ｶｸﾄﾞ  :? 60
CSEC( 60°)=   1.15470
Ok
```

- 角度を度で読み込んで，コセカント(CSEC)の値を出力するプログラムです.
- 150行で

$$\mathrm{csec}\,\theta = 1/\sin\theta$$

を用いて計算しています.

参照：COS, TAN, ATN

SOUND

サウンド：sound

機　能　PSGのレジスタにデータをセットする

書　式　SOUND〈レジスタ番号〉,〈数式〉

解　説
- このステートメントは本体にミュージックインタフェースボード(PC-8801-10)が接続してあるときに使用します.
- SOUND文は，ミュージックインタフェースボードに実装されている2個のPSGレジスタに直接データを書き込みます.
- 〈数式〉は0〜255までの整数で指定します.
- 2個のPSGにはそれぞれ16個のレジスタがあり，次のような機能を持っています.

SOUNDレジスタ表

PSG1 レジスタ	PSG2 レジスタ	ビット / 機能	MSB B7	B6	B5	B4	B3	B2	B1	LSB B0
R 0	R16	チャネルA周波数	8BIT FT-A							
R 1	R17						4BIT CT-A			
R 2	R18	チャネルB周波数	8BIT FT-B							
R 3	R19						4BIT CT-B			
R 4	R20	チャネルC周波数	8BIT FT-C							
R 5	R21						4BIT CT-C			
R 6	R22	ノイズ周波数				5BIT NP				
R 7	R23	チャネル設定	$\overline{I/O}$		$\overline{NOISE}$			$\overline{TONE}$		
					C	B	A	C	B	A
R 8	R24	チャネルA音量				M	L3	L2	L1	L0
R 9	R25	チャネルB音量				M	L3	L2	L1	L0
R10	R26	チャネルC音量				M	L3	L2	L1	L0
R11	R27	エンベロープ周期	8BIT FT							
R12	R28		8BIT CT							
R13	R29	エンベロープ形状					E3	E2	E1	E0
R14	R30	I/OポートAデータ	8BITパラレルI/OポートA							
R15	R31	I/OポートBデータ	8BITパラレルI/OポートB							

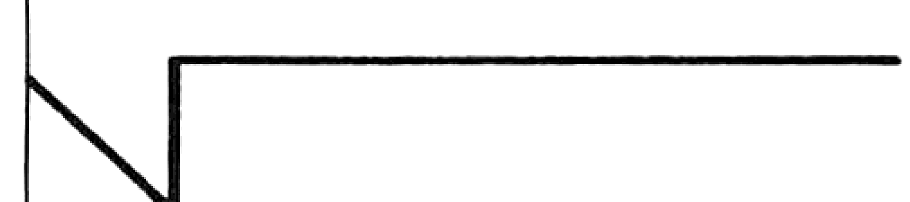
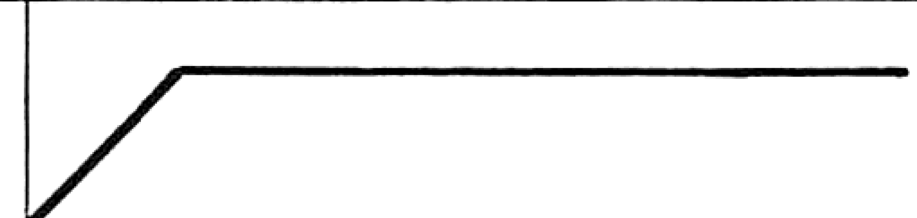
FTは微調整，CTは主調整，NPは周波数調整を意味します．

R7, R23の第6，7ビットおよびR14, R15, R30, R31は，音を出すには直接関係ありません(I/Oポートの機能は無効です)．

R8, R9, R10, R24, R25, R26は，エンベロープ・モードにすると音量の設定はできません．

• エンベロープの形状には次のようなものがあります．

エンベロープ形状

R13, R29の値	エンベロープパターン
0, 1, 2, 3, 9	
4, 5, 6, 7, 15	
8	
10	
11	
12	
13	
14	

• 詳しくは，ミュージックインタフェースボードに添付されているユーザーズマニュアルを参照してください．

例

SOUND 7,0

• ミュージックインタフェースボードのPSG1のレジスタ7に0を書き込みます．

プログラム例

このプログラムを実行する場合には，別売のミュージックインタフェースボードが必要です．

```
list
100 ' SOUND
110 STOPM
120 SOUND 7,7
130 SOUND 23,7
140 SOUND 6,30
150 SOUND 22,30
160 SOUND 13,12
170 SOUND 29,12
180 FOR I=1 TO 30
190   FOR L=0 TO 15
200     SOUND 8,L
210   NEXT L
220   FOR L=15 TO 0 STEP -3
230     SOUND 8,L
240   NEXT L
250 NEXT I
260 END
Ok
```

• 蒸気機関車が走る音を出します．

• 120行と130行でノイズの設定を，140行と150行ではノイズの周波数をコントロールしています．

• 160行及び170行ではエンベロープの形状を設定しており，200行と230行で音を出しています．

注意： • 本体にミュージックインタフェースボードを接続して使用する場合，ミュージックインタフェースボードのジャンパスイッチのジャンパコネクタは取り外してください．

SPACE$

スペース・ダラー：space $

機能 指定した長さの空白文字列を返す

書式 SPACE$(〈数式〉)

解説 ・〈数式〉の数だけの空白を持った文字列を返します．〈数式〉の値の範囲は0～255です．

例 SPACE$(7) 返される値 "□□□□□□□"

・7個の空白を持った文字列を返します．

プログラム例

```
list
100 ' SPACE$ sample
110 WIDTH 80,25
120 FOR I=-6 TO 6
130   A$=CHR$(&H88)+SPACE$(I^2)+"*"
140 PRINT A$
150 NEXT I
160 END
Ok
run
|                                    *
|                         *
|                *
|         *
|    *
| *
|*
| *
|    *
|         *
|                *
|                         *
|                                    *
Ok
```

・"＊"で放物線を描くプログラムです．

・130行で

"┃"＋"Ｉの2乗個の空白"＋"＊"

を変数A$に代入しています．Ｉは－6から6まで変化するため，変数A$を順に出力すると，"＊"が放物線を描いているように見えます．

参照：SPC

SPC

エス・ピー・シー：space

機能 指定した数だけ空白を出力する

書式 SPC(〈数式〉)

解説
- SPC関数は，PRINT文などの出力文中でのみ使用することができます．
- 〈数式〉の値の範囲は0〜255です．

例 SPC(7) 返される値 "□□□□□□□"

- ただし，PRINT文などの出力文中で使用します．

プログラム例

```
list
100 ' SPC sample
110 WIDTH 80,25
120 FOR I=-6 TO 6
130 PRINT CHR$(&H88);SPC(I^2);"*"
140 NEXT I
150 END
Ok
```

- SPACE$関数のプログラム例と同じ動作をします．
- SPACE$関数の例では，連結した文字列をA$に代入してから出力していましたが，ここでは"|"と空白と"*"を順に直接画面に出力しています．

参照：SPACE$, TAB

SQR

スクウェア・ルート：square root

機能 平方根(スクウェアルート)を返す

書式 SQR(〈数式〉)

解説
- 〈数式〉で指定された値の平方根を返します.
- SQR関数の演算は単精度で行われます.
- 〈数式〉の値が負のときは "Illegal function call" エラーとなります.

例

SQR(2) 返される値 1.41421

SQR(−2) 返される値 "Illegal function call" エラー

プログラム例

```
list
100 ' SQR sample
110 PRINT " << SQUARE ROOT TABLE >>"
120 FOR N=0 TO 10
130   PRINT "N =";N,"ROOT(N) =";
140   PRINT SQR(N)
150 NEXT N
160 END
Ok
run
 << SQUARE ROOT TABLE >>
N = 0          ROOT(N) = 0
N = 1          ROOT(N) = 1
N = 2          ROOT(N) = 1.41421
N = 3          ROOT(N) = 1.73205
N = 4          ROOT(N) = 2
N = 5          ROOT(N) = 2.23607
N = 6          ROOT(N) = 2.44949
N = 7          ROOT(N) = 2.64575
N = 8          ROOT(N) = 2.82843
N = 9          ROOT(N) = 3
N = 10         ROOT(N) = 3.16228
Ok
```

- 0から10までの平方根を出力します.

STOP

ストップ：stop

機能 プログラムの実行を停止してコマンドレベルに戻る

書式 STOP

解説 • STOP文はプログラムの実行を停止するもので，プログラム中のどこに置いてもかまいません．

• STOP文を実行すると，次のメッセージが表示され，コマンドレベルに戻ります．

Break in xxxxx

(xxxxxは停止した行の行番号)

• END文と異なり，STOP文はファイルを閉じません．

• CONT文により，停止したSTOP文の次からプログラムの実行を再開することができます．

例 STOP

• プログラムの実行を停止させます．

プログラム例

```
list
100 ' STOP sample
110 I=10
120 PRINT I,SQR(I)
130 STOP
140 I=I+1
150 GOTO 120
Ok
run
 10             3.16228
Break in 130
Ok
```

• このプログラム例は，130行がなければ10以上の整数とその平方根を次々と出力するものです．

• 実行されると，10とその平方根を出力してすぐ停止します．STOP文による停止なので，CONT文を入力することによってSTOP文の次の140行から実行を続けることができます．

参照：CONT

STOPM

ストップ・エム：stop music

機能 初期設定を行う

書式 STOPM

解説
・ボリューム，オクターブ等の音楽機能の初期化を行います．

・PLAY文，SOUND文，VOICE REG文で発生させた音で，Key-on中(鍵盤楽器にたとえれば，鍵盤を押した状態)の音はSTOPM文で止めることができます．ただし，Release Rateの設定値によっては音が残るものがあります．

例 STOPM

・初期設定を行います．

プログラム例

```
list
100 ' STOPM sample
110 BGM 0:STOPM
120 PLAY "@605T80L1"
130 PLAY "V15C&C"
140 FOR W=0 TO 500:NEXT W
150 STOPM
160 PLAY "V15C&C"
170 END
Ok
```

・120行で音色をパイプオルガンにして，130行で音を出します．150行でSTOPMを実行すると，160行ではオクターブ4，テンポ120，音符の長さ4(4分音符)のパイプオルガンの音で演奏されます．

STR$

エス・ティー・アール・ダラー：string $

機能 数値を文字列に変換した値を返す

書式 STR$(〈数式〉)

解説
- 〈数式〉によって指定された値を文字列に変換した値を返します．〈数式〉の値はすべての数値型を使うことができます．
- 変換された文字列の最初のキャラクタは符号です．(正の数の場合は空白，負の数の場合は"−"となります.)

例

STR$(123) 返される値 " 123"

STR$(−1.5E+10) 返される値 "−1.5E+10"

プログラム例

```
list
100 ' STR$ sample
110 INPUT A#:IF A#<0# OR 1D+16=<A# THEN 110
120 A$=STR$(A#):L=LEN(A$)-1
130 K=L MOD 3:IF K=0 THEN K=3
140 B$=LEFT$(A$,K+1):A$=MID$(A$,K+2)
150 IF A$="" THEN 190
160 B$=B$+","+LEFT$(A$,3)
170 A$=MID$(A$,4)
180 GOTO 150
190 PRINT B$
200 END
Ok
run
? 123456789
 123,456,789
Ok
run
? 123
 123
Ok
run
? 12345678901234567
? 1234567890123456
 1,234,567,890,123,456
Ok
```

- 正整数を読み込んで，3桁ごとに","を付けて出力します．
- 倍精度実数は16桁までは指数形式を用いないで表示できます．したがって，読み込んだ数値が16桁を超えた場合は，110行でもう一度読み込むようにしています．
- 120行で読み込んだ数値を文字列に変換しています．

参照：VAL

STRING$

ストリング・ダラー：string $

機　能　指定した文字からなる指定した長さの文字列を返す

書　式　STRING$(〈式〉, |〈文字列〉|)
　　　　　　　　　　　　　 |〈数式〉|

解　説　•与える文字は，〈文字列〉または〈数式〉によって指定します．〈文字列〉の場合，最初の１文字が有効です．〈数式〉の場合はキャラクタコード(0～255の範囲の値)で指定します．

例　STRING$(5, "＊")　返される値 ⇒　"＊＊＊＊＊"

STRING$(2, 75)　返される値 ⇒　"KK"

プログラム例

```
list
100 ' STRING$ sample
110 INPUT "ﾓｼﾞﾚﾂ ｦ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ";A$
120 L=LEN(A$)
130 PRINT STRING$(L+4,"+")
140 PRINT "+ ";SPC(L+1);"+"
150 PRINT "+ ";A$;" +"
160 PRINT "+ ";SPC(L+1);"+"
170 PRINT STRING$(L+4,"+")
180 END
Ok
run
ﾓｼﾞﾚﾂ ｦ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ? STRING$ sample program
++++++++++++++++++++++++++
+                        +
+ STRING$ sample program +
+                        +
++++++++++++++++++++++++++
Ok
```

•読み込んだ文字列を"＋"で囲んで出力するプログラムです．

参照：資料８　キャラクタコード

SWAP

スワップ：swap

機　能　2つの変数の値を交換する

書　式　SWAP〈変数名1〉,〈変数名2〉

解　説　• どの型の変数(整数,単精度,倍精度,文字,単純変数と配列変数にかかわらず)でも，SWAP文によりその値を入れ替えることができます．ただし，その2つの変数の型は一致していなければなりません．

• 〈変数名2〉が単純変数(配列変数以外の変数)のとき，その変数には値が定義されていなければなりません．〈変数名1〉の変数は，定義されていなくてもかまいません．

例　SWAP A$,B$

• A$とB$に入っている値を入れ替えます．

プログラム例

```
list
100 ' SWAP sample
110 A$="ﾀﾛｰ":B$="ﾊﾅｺ":C$="ｱｲｼﾃｲﾏｽ":D$="ｷﾗｯﾃｲﾏｽ"
120 GOSUB 160
130 SWAP A$,B$:SWAP C$,D$
140 GOSUB 160
150 END
160 PRINT USING "& & ﾊ & & ｦ &      &";A$,B$,C$
170 RETURN
Ok
run
ﾀﾛｰ ﾊ ﾊﾅｺ ｦ ｱｲｼﾃｲﾏｽ
ﾊﾅｺ ﾊ ﾀﾛｰ ｦ ｷﾗｯﾃｲﾏｽ
Ok
```

• "タロー　ハ　ハナコ　ヲ　アイシテイマス"と，"ハナコ　ハ　タロー　ヲ　キラッテイマス"という文を出力するプログラムです．

• 130行で，"タロー"と"ハナコ"，"アイシテイマス"と"キラッテイマス"をそれぞれ入れ替えています．

TAB

タブ：tab

機能 指定された桁位置まで空白を出力する

書式 TAB(〈数式〉)

解説
- TAB関数はPRINT文などの出力文中のみで使用されます．
- 〈数式〉の値の範囲は0～255です．
- 〈数式〉の値が1行当たりの桁数よりも大きくなったときは自動的に改行されます．

例 TAB(20) 返される値 (20桁目までの空白)

- ただし，PRINT文などの出力文中で使用します．

プログラム例

```
list
100 ' TAB sample
110 FOR DEG=0 TO 180 STEP 20
120   RAD=DEG/180*3.14159
130   PRINT "DEG =";DEG;
140   PRINT TAB(15)"SIN(DEG) =";SIN(RAD);
150   PRINT TAB(40)"COS(DEG) =";COS(RAD)
160 NEXT DEG
170 END
Ok
run
DEG = 0        SIN(DEG) = 0            COS(DEG) = 1
DEG = 20       SIN(DEG) = .34202       COS(DEG) = .939693
DEG = 40       SIN(DEG) = .642787      COS(DEG) = .766045
DEG = 60       SIN(DEG) = .866025      COS(DEG) = .500001
DEG = 80       SIN(DEG) = .984808      COS(DEG) = .173649
DEG = 100      SIN(DEG) = .984808      COS(DEG) =-.173647
DEG = 120      SIN(DEG) = .866026      COS(DEG) =-.499999
DEG = 140      SIN(DEG) = .642789      COS(DEG) =-.766043
DEG = 160      SIN(DEG) = .342023      COS(DEG) =-.939692
DEG = 180      SIN(DEG) = 2.8088E-06   COS(DEG) =-1
Ok
```

- 角度と，その正弦と余弦を出力するプログラムです．
- 140行と150行のTAB関数で，"SIN"と"COS"をそれぞれ15桁目と40桁目にそろえて出力させています．

参照：SPC

T

TAN

タンジェント：tangent

機能 正接(タンジェント)を返す

書式 TAN(〈数式〉)

解説
- 〈数式〉の値の正接(タンジェント)を返します.
- 〈数式〉の値の単位はラジアン(π/180×角度)です.
- TAN関数の演算は単精度で行われます.

例

TAN(3.14159/180*30) 返される値 0.57735(TAN(30°)の値)

TAN(3.14159/180*60) 返される値 1.73205(TAN(60°)の値)

プログラム例

```
list↵
100 ' TAN sample
110 F$="COT(###°)=###.#####
120 INPUT "ｶｸﾄﾞ :";D
130 R=D MOD 180
140 IF R=0 THEN PRINT "ﾃｲｷﾞｻﾚﾏｾﾝ":END
150 C=1/TAN(R/180*3.1416)
160 PRINT USING F$;D,C
170 END
Ok
run↵
ｶｸﾄﾞ :? 45↵
COT( 45°)=  1.00000
Ok
run↵
ｶｸﾄﾞ :? 30↵
COT( 30°)=  1.73205
Ok
```

- 角度を度で読み込んでコタンジェント(COT)を求めるプログラムです.
- 150行で

  $\cot\theta = 1/\tan\theta$

を使って求めています.

注意：• TAN関数は

TAN(〈数式〉)＝SIN(〈数式〉)/COS(〈数式〉)

という式で算出しているため，COS関数の値が0となるとき(つまり，〈数式〉の値がπ/2, 3π/2, 5π/2, ...のとき)"Division by zero" エラーとなります.

参照：SIN, COS, ATN

TIME$

タイム・ダラー：time $

機能 内蔵のカレンダ時計の時刻を返す

書式 TIME$

解説

• TIME$変数を参照することにより内蔵カレンダ時計の時刻が得られます．

• 時刻は8文字の文字列で返され，文字変数に代入したり，PRINT文で内容を表示することができます．

• 時刻を設定するには，TIME$変数に次の形式の文字列を代入します．

"HH：MM：SS"

HH ･･･時(00～23)
MM･･･分(00～59)
SS ･･･秒(00～59)

各2文字で指定します．

例

TIME$ 返される値 (現在の時刻)

TIME$＝"21：11：00"

• 内蔵カレンダ時計に21時11分を設定します．

プログラム例

```
list
100 ' TIME$ sample
110 WIDTH 40,20:CLS
120 LOCATE 13,7
130 PRINT "ﾀﾀﾞｲﾏ ﾉ ｼﾞｺｸ"
140 LOCATE 15,9
150 PRINT TIME$
160 GOTO 140
Ok
```

• 画面上に時刻を表示させます．

参照：DATE$

TRON/TROFF

トレース・オン/トレース・オフ：trace on/trace off

機能 実行トレースモードを制御する

書式 TRON

TROFF

解説 • プログラムのデバッグのために実行トレースモードがあります．トレースモードでプログラムを実行すると，実行されている行の行番号が表示されます．

• TRONコマンドで実行トレースモードに入ります．TROFFコマンドは，それを解除するコマンドです．NEWコマンドを実行した場合も，実行トレースモードは解除されます．

例 TRON

• 実行トレースモードに入ります．

実　行　例

```
list
100 ' TRON/TROFF sample
110 I=1
120 PRINT I
130 I=I+1
140 IF I<=3 THEN 120
150 END
Ok
tron
Ok
run
[100][110][120] 1
[130][140][120] 2
[130][140][120] 3
[130][140][150]
Ok
troff
Ok
run
 1
 2
 3
Ok
```

• TRONコマンドを使って，上記のプログラムの実行されている行番号を表示しています．TROFFコマンドにより，通常の実行状態に戻っています．

USR

ユーザ：user

機　能　機械語で作られたユーザ関数ルーチンを呼び出し，その値を返す

書　式　USR〔〈番号〉〕(〈引数〉)

解　説

- USR関数は，ユーザの作成した機械語関数ルーチンを呼び出します．
- USR関数は，DEF USR文で実行開始番地を設定した上で使用します．
- 〈番号〉は0〜9の値で，DEF USR文により定義された番号に対応します．〈番号〉が省略された場合は0と見なされます．
- 〈引数〉によりBASICから機械語ルーチンへの値の受け渡しをすることができます．
- USR関数の詳しい使い方は，3章　機械語プログラムの呼び出し方を参照してください．

例　USR3(0) 返される値 ⇒ (USR3関数を呼び出して返される値)

プログラム例

```
list
100 ' USR sample
110 CLEAR 300,&HBFFF
120 DEF USR=&HC000:GOSUB 210
130 CONSOLE 0,25,0,1:WIDTH 80,25:CLS
140 FOR I=1 TO 80*24
150   PRINT "♥";
160 NEXT I
170 CLS:FOR I=1 TO 500:NEXT I
180 VAD%=&HF3C8
190 I=USR(VAD%):FOR I=1 TO 500:NEXT I
200 END
210 FOR AD=&HC000 TO &HC015
220   READ DA:POKE AD,DA
230 NEXT AD
240 RETURN
250 DATA &H7E            :'      LD   A,(HL)
260 DATA &H23            :'      INC  HL
270 DATA &H66            :'      LD   H,(HL)
280 DATA &H6F            :'      LD   L,A
290 DATA &H3E,&HE9       :'      LD   A,"♥"
300 DATA &H11,&H28,&H00  :'      LD   DE,40
310 DATA &H0E,&H19       :'      LD   C,25
320 DATA &H06,&H50       :' L1:  LD   B,80
330 DATA &H77            :' L2:  LD   (HL),A
340 DATA &H23            :'      INC  HL
350 DATA &H10,&HFC       :'      DJNZ L2
360 DATA &H19            :'      ADD  HL,DE
370 DATA &H0D            :'      DEC  C
380 DATA &H20,&HF6       :'      JR   NZ,L1
390 DATA &HC9            :'      RET
Ok
```

- BASICと機械語のプログラムでテキスト画面全体に"♥"を2回描きます．

U

• このプログラム例はDEF USR文と同じものです.

参照：DEF USR

VAL

バリュー：value

機能　文字列の表す数値を返す

書式　VAL(〈文字列〉)

解説
- 〈文字列〉を数値に変換した値を返します.
- 〈文字列〉の最初の文字が"+","−","&","."，または数字でなければVAL関数の値は0になります．また，数字以外の文字(ただし，16進数の場合の"A"〜"F"，&Hの"H"，および指数表現の"D"と"E"を除く)が含まれている場合，それ以降の文字は無視されます.
- 文字列中のスペースは無視します.

例

VAL("123") 返される値 ⇒ 123

VAL("&HFF") 返される値 ⇒ 255(=&HFF)

プログラム例

```
list
100 ' VAL sample
110 T$=TIME$
120 H=VAL(LEFT$(T$,2))
130 M=VAL(MID$(T$,4,2))
140 S=VAL(RIGHT$(T$,2))
150 SS=H*3600+M*60+S
160 PRINT "00:00:00 ｶﾗ ";T$;" ﾏﾃﾞﾊ";
170 PRINT SS;"秒 ﾃﾞｽ"
180 END
Ok
run
00:00:00 ｶﾗ 15:12:04 ﾏﾃﾞﾊ 54724 秒 ﾃﾞｽ
Ok
```

- 00：00：00から現在の TIME$ 変数の時刻までの 秒数を出力するプログラムです.
- 120行から140行で，時，分，秒を表す文字列を数値に変換しています.

参照：STR$

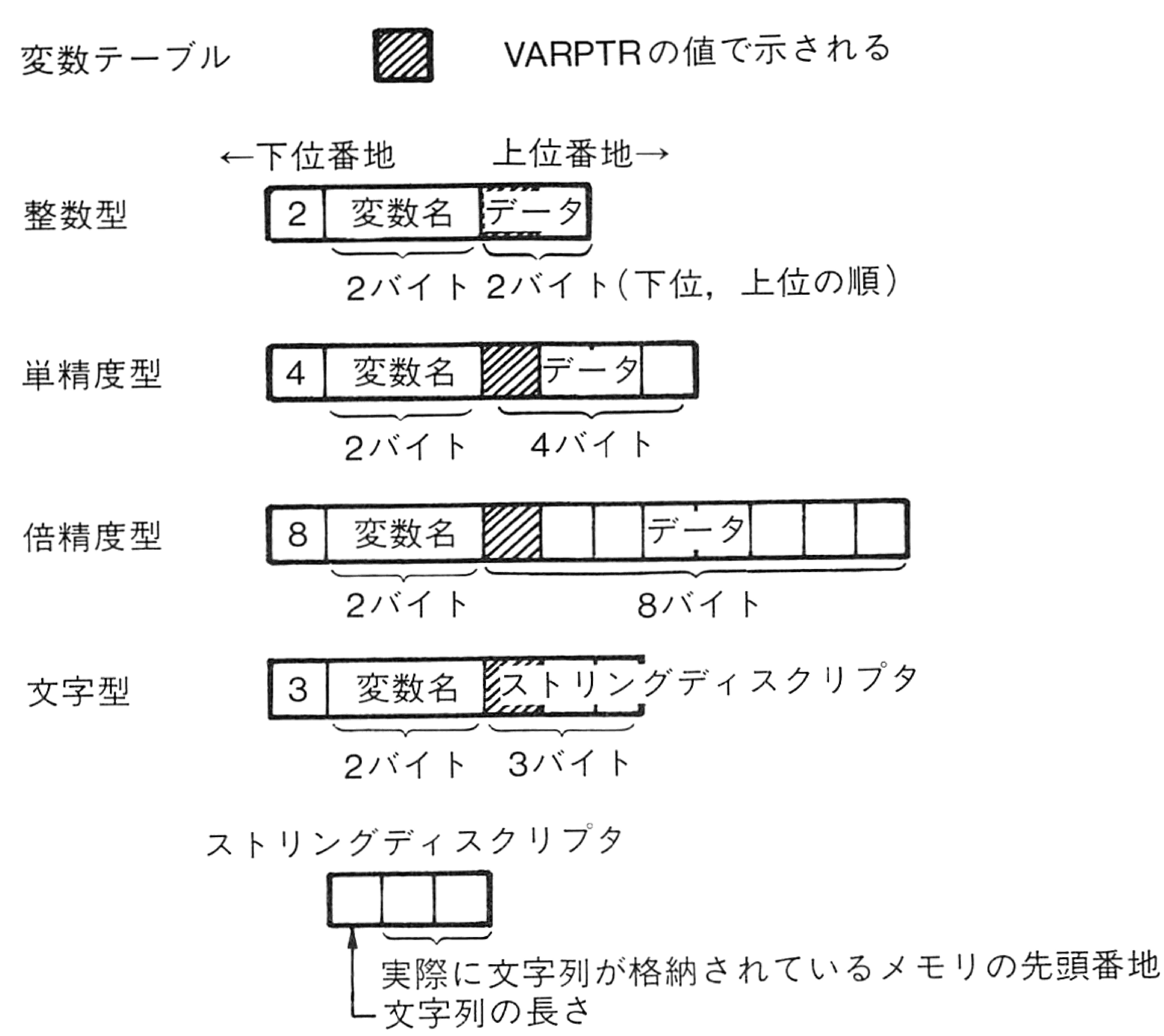
VARPTR

バリアブル・ポインタ：variable pointer

機能 変数の格納されているメモリ番地，ファイルに割り当てられているバッファの先頭番地を返す

書式 VARPTR (〈変数名〉)
VARPTR (#〈ファイル番号〉)

解説 •引数として〈変数名〉を指定すると，その変数のデータが格納されている変数領域のメモリ番地を返します．このとき指定される変数は値が代入されていなければなりません．



•引数として〈ファイル番号〉を指定すると，その〈ファイル番号〉に割り当てられている入出力バッファの先頭番地－9を返します．

例 VARPTR(#1) 返される値 (入出力バッファ1の先頭番地－9)

プログラム例

```
list
100 ' VARPTR sample
110 DEFINT A
120 INPUT "Value";A
```

```
130 AD=VARPTR(A)
140 AL=(A/2) MOD 256
150 AH=(A¥2)¥256
160 POKE AD,AL
170 POKE AD+1,AH
180 PRINT A
190 GOTO 120
Ok
run⏎
Value? 1245⏎
 622
Value? 510⏎
 255
Value? 9035⏎
 4517
Value? STOP
Break in 120
Ok
```

• 整数値を読み込んで，その½の値を出力します．小数点以下は切り捨てます．

• 130行で，実数Aの値が格納されているアドレスを変数ADに代入します．

• 140行と150行で上位バイトと下位バイトを別々に求めて，160行と170行で直接値を書き替えています．

VIEW

ビュー・view

機能 グラフィック画面上でのビューポートを設定する

書式 VIEW(gx1, gy1)−(gx2, gy2)

解説

• (gx1, gy1), (gx2, gy2)を対角とする長方形の枠内をグラフィック領域として指定します．グラフィック領域とは，グラフィック命令の操作対象となる座標系のことで，ビューポートと呼びます．

• VIEW文はグラフィック領域の指定を行うだけで，実際の画面に対する操作は行いません．

• gx1＜gx2, gy1＜gy2が成り立たない場合，あるいはこれらの座標が次の範囲にないときは "Illegal function call" エラーとなります．

0≦gx1, gx2≦ 639(640×200ドットモード) / 319(320×200ドットモード)

0≦gy1, gy2≦199

・LPは，ビューポートの左上の頂点(gx1, gy1)に設定されます．

・システム起動時，SCREEN文(第一パラメータが設定されている)実行後のビューポートは次のように設定されています．

640×200ドットモード(SCREEN 0又はSCREEN 1)

VIEW(0, 0)−(639, 199)

320×200ドットモード(SCREEN 2又はSCREEN 3)

VIEW(0, 0)−(319, 199)

• CLS 2または3を実行した場合，グラフィック画面はビューポート内のみが消去されます．

例 VIEW(0, 0)−(99, 99)

• ビューポートを(0, 0)〜(99, 99)に設定します．

プログラム例

```
list
100 ' VIEW sample
110 SCREEN 0,0:CLS 3
120 FOR I=0 TO 100
130   X1=RND(1)*640:X2=RND(1)*640:IF X1>=X2 THEN 130
140   Y1=RND(1)*200:Y2=RND(1)*200:IF Y1>=Y2 THEN 140
150   COLOR ,RND(1)*7+1
160   VIEW(X1,Y1)-(X2,Y2)
170   CLS 2
180 NEXT I
```

```
190 END
Ok
```

• いろいろな四角形を描きます．

• 130行から140行で乱数を使ってビューポートの座標を求め，150行でバックグラウンドカラーを設定します．

• 160行で，130行および140行で求めた座標でビューポートを設定します．

• 170行で，150行で指定されたバックグラウンドカラーでビューポート内を消去します．つまり，ビューポート内をバックグラウンドカラーで塗りつぶすことによって四角形が描かれることになります．

参照：SCREEN，CLS

VIEW

ビュー：view

機能 ビューポートの設定位置を返す

書式 VIEW(〈機能〉)

解説 • VIEW文により設定されているビューポートの位置(gx1, gy1)−(gx2, gy2)を返します.

• 〈機能〉は0〜3の整数値で指定します.

0：ビューポート左上の頂点のX座標(gx1)

1：ビューポート左上の頂点のY座標(gy1)

2：ビューポート右下の頂点のX座標(gx2)

3：ビューポート右下の頂点のY座標(gy2)

例 VIEW(0) 返される値 (ビューポート左上の頂点のX座標値)

プログラム例

```
list
100 ' VIEW sample
110 VIEW(10,20)-(30,40)
120 IF VIEW(0)=10 AND VIEW(1)=20 AND VIEW(2)=30 AND VIEW(3)=40
    THEN PRINT "Good VIEW function"
130 END
Ok
```

• VIEW(10, 20)−(30, 40)でビューポートを設定したあと, VIEW関数で読み込んだ座標と指定した座標が一致していたら "Good VIEW function" を出力します.

参照：VIEW

VOICE

ボイス：voice

機　能　音色を設定する

書　式　VOICE〈整数配列名1〉〔,〈整数配列名2〉〕〔,〈整数配列名3〉〕

解　説

• 指定された配列変数の値を音色設定のパラメータとしてFM音源に送り，音色を設定します．

• 整数配列名1〜3は，それぞれチャネル1〜3に対応します．

• 配列変数はDIM文で定義し，添字は(4, 9)で宣言します．

• 各配列変数の意味は次のようになります．ここでは配列変数名をV%として説明しています．

V%(0, 0): Feedback(フィードバック)/Algorithm(アルゴリズム)

これは6ビットのデータになっていて,以下の意味を持っています．

0	0	D5	D4	D3	D2	D1	D0
		Feedback			Algorithm		

使用例) V%(0,0)=8*FB+AR

FBはFeedbackを意味し，0〜7の値を指定します．

ARはAlgorithmを意味し，0〜7の値を指定します．

Feedback

第1オペレータの変調度を決めます．指定できる範囲は0〜7までで，大きく指定するほど金属的な音になります．

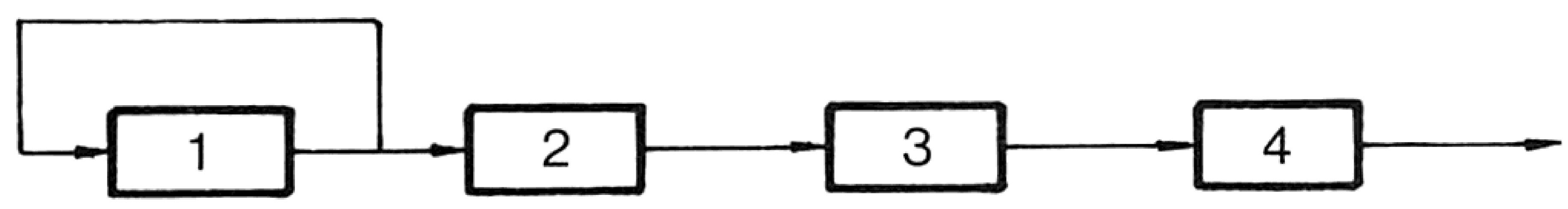
第1オペレータは自分自身の出力を変調信号としているため，この変調度をこれにより制御します．

値	0	1	2	3	4	5	6	7
変調度	OFF	$\pi/16$	$\pi/8$	$\pi/4$	$\pi/2$	π	2π	4π

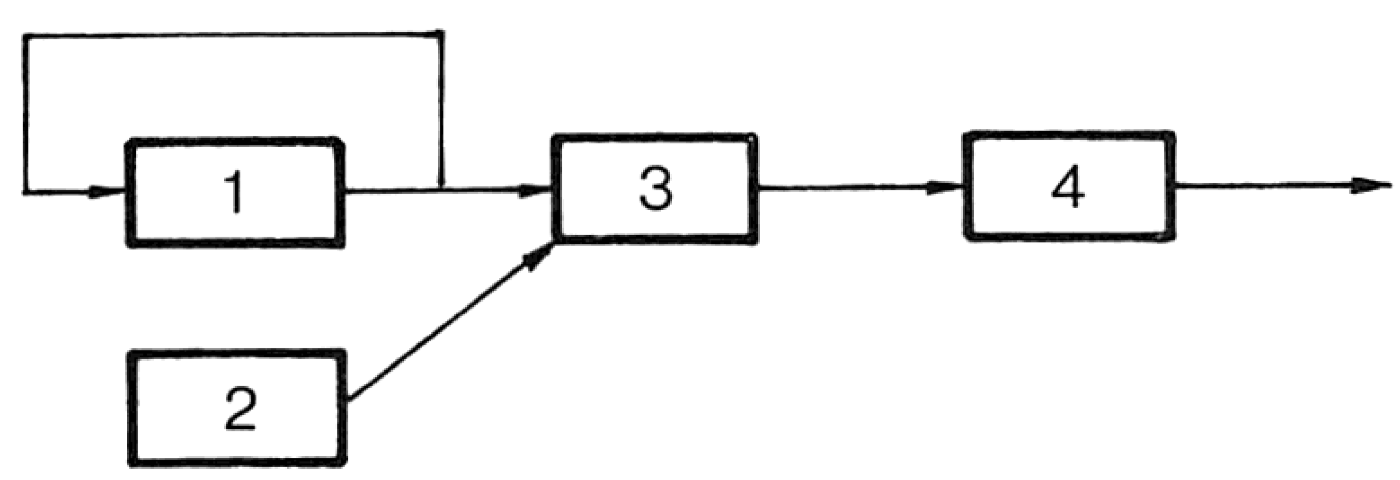
Algorithm

4つのオペレータの接続の仕方を決定します．指定できる範囲は0〜7までで，それぞれ次のようなアルゴリズムを選択します．

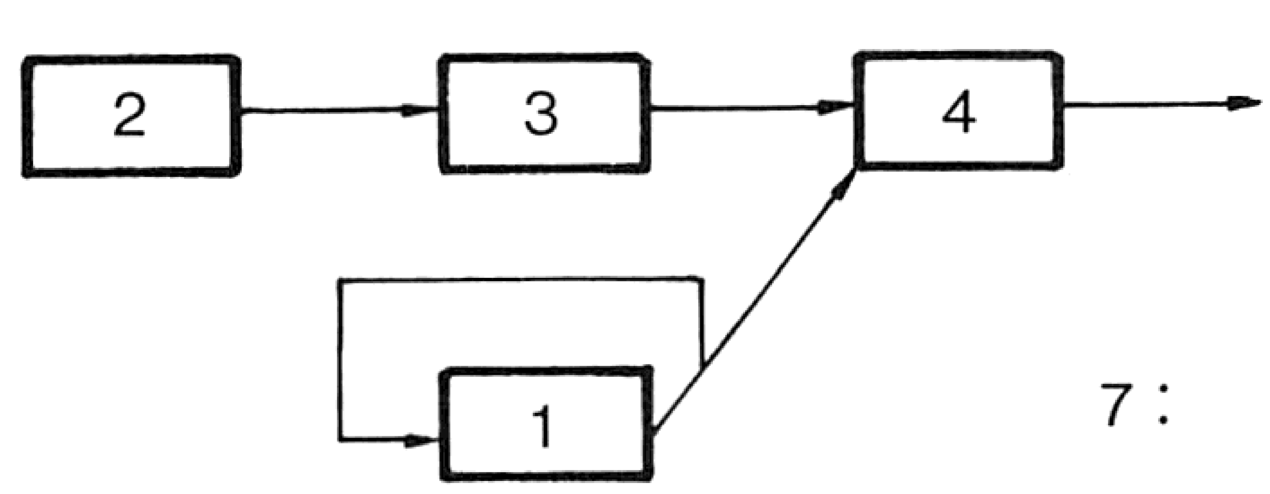
0:



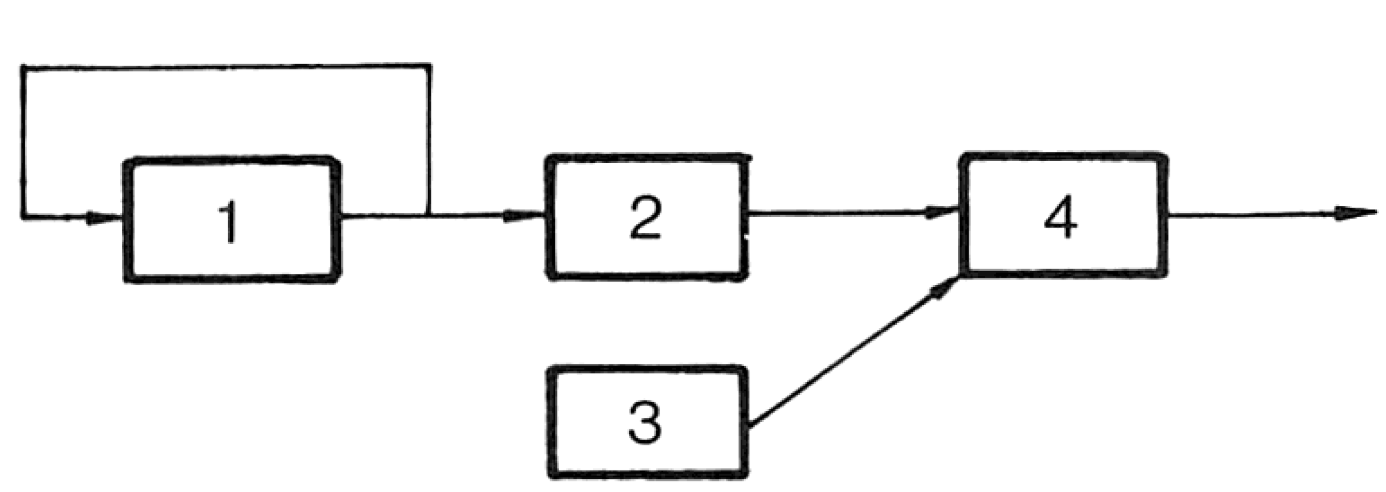
1:



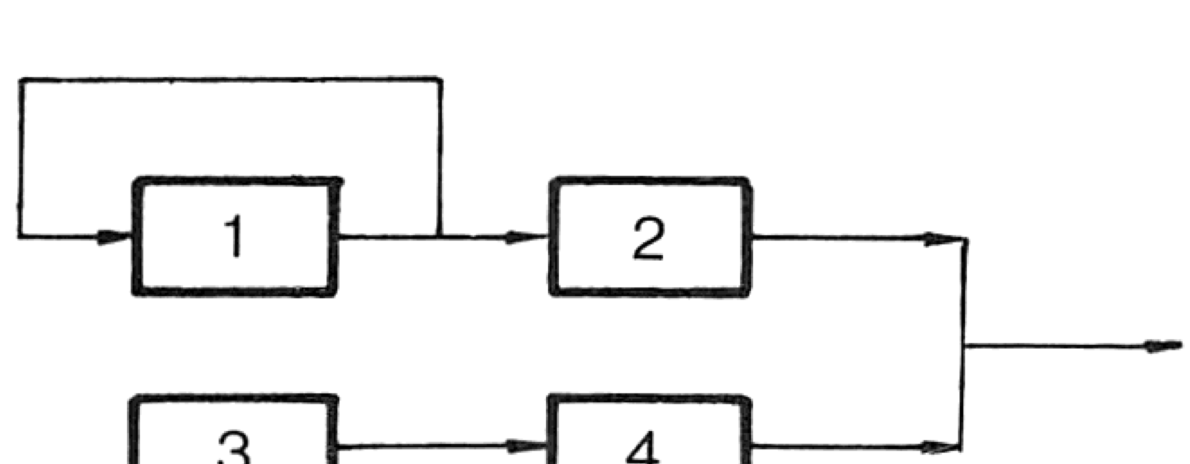
2:



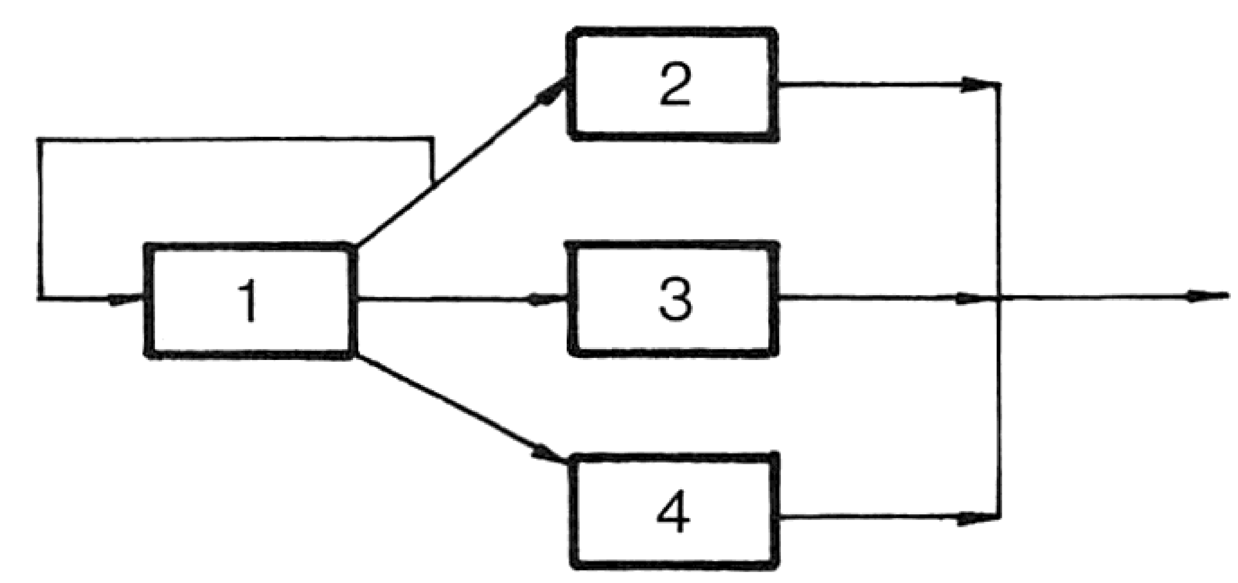
3:



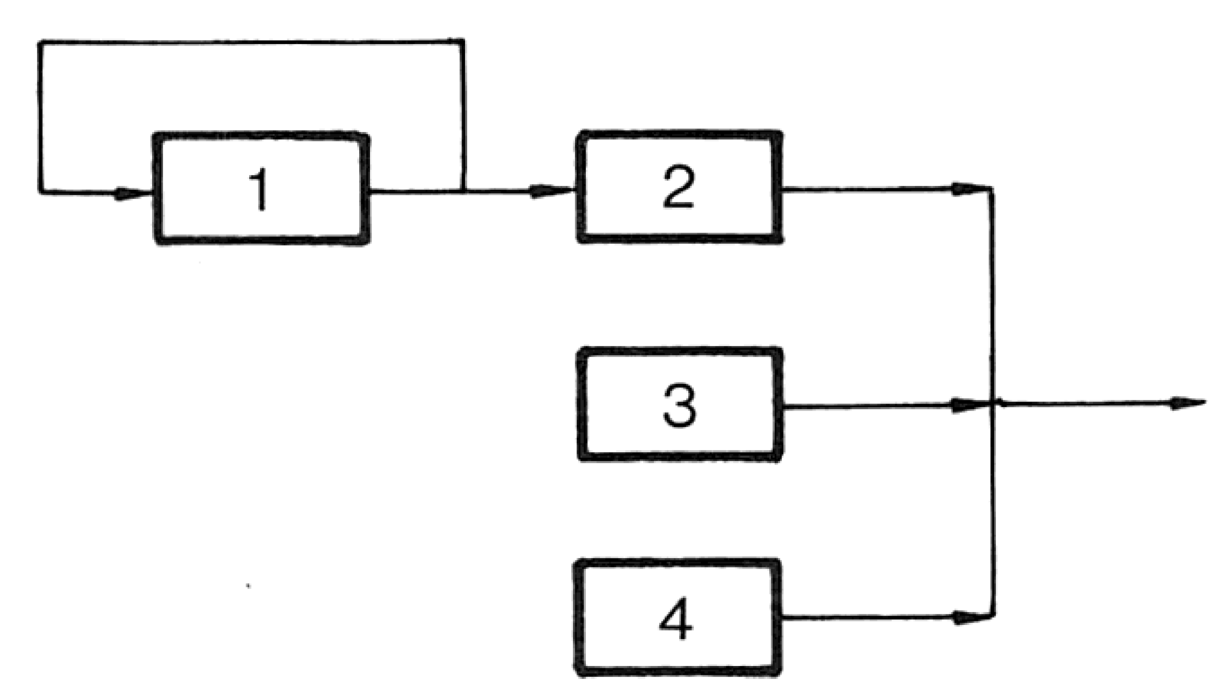
4:



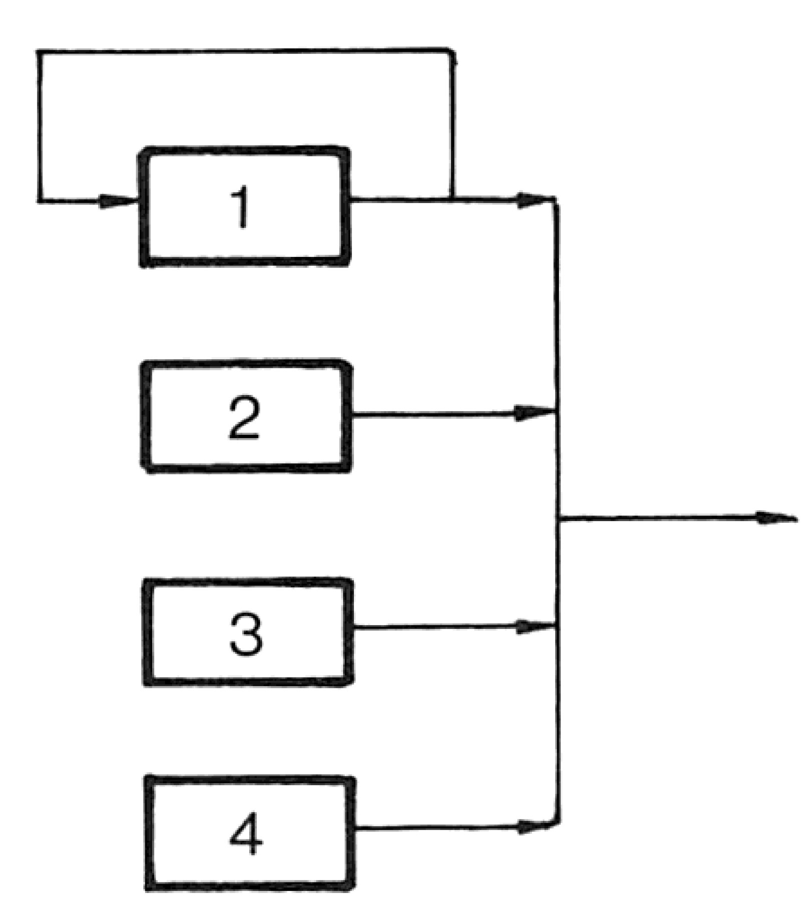
5:



6:



7:

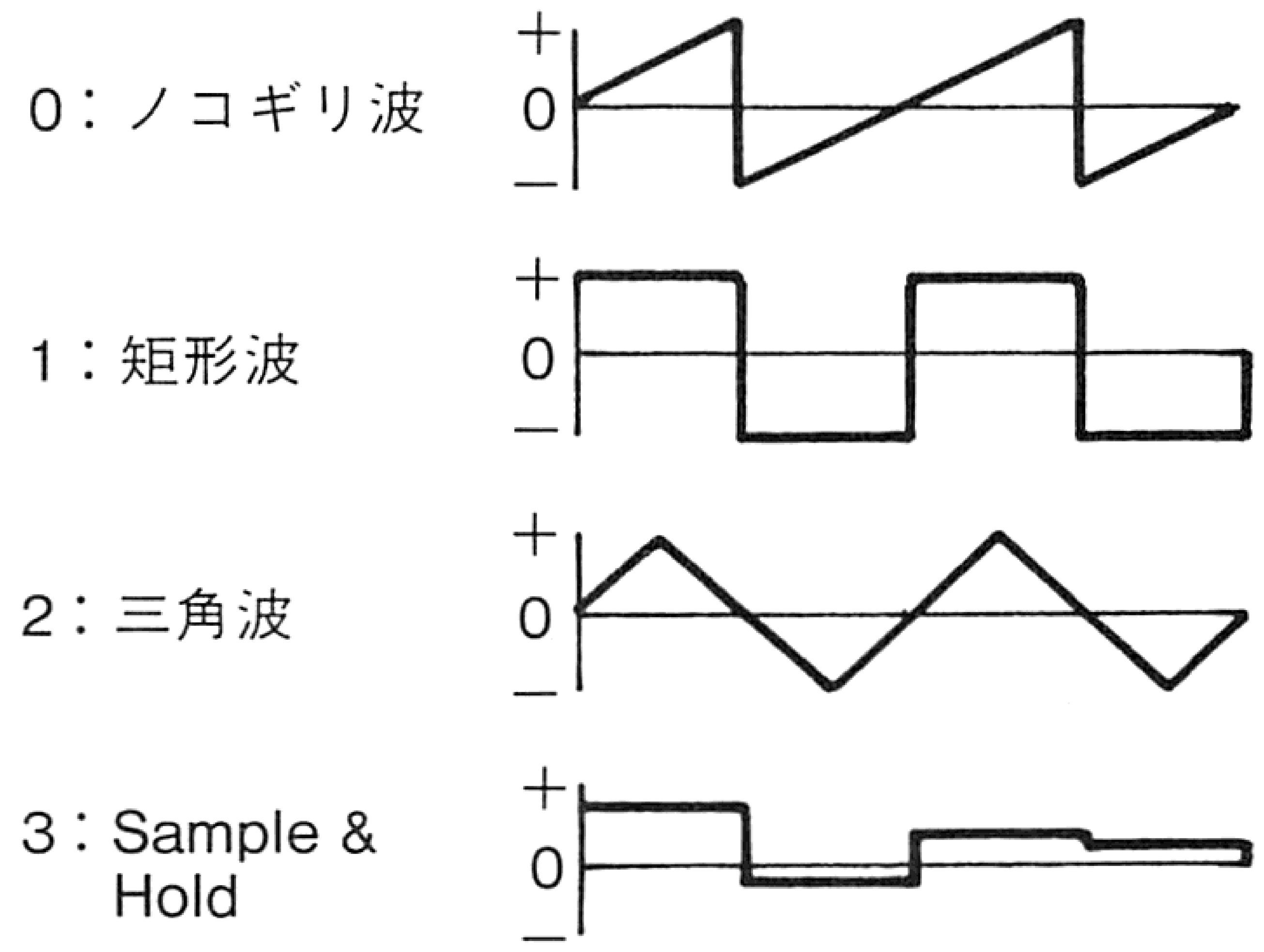


V%(0, 1): Operator mask(オペレータマスク)

各オペレータを使用するか否かを決定します．0でオペレータをOFF，1でオペレータをONにします．それぞれの値は次のとおりです．

オペレータ				値
OP4	OP3	OP2	OP1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

V%(0, 2): Wave Form(ウェーブフォーム)



Sample & Hold(サンプルアンドホールド)はランダム値が出力されます．

V%(0, 3): Sync(シンク)

0: Key-on(鍵盤楽器にたとえれば，鍵盤を押した状態)と無関係に

LOF(Low Frequency Oscillator)が動作します.

1: Key-off(鍵盤楽器にたとえれば，鍵盤から指を離した状態)と同期してLFOが動作します.

V%(0, 4): Speed(スピード)

LFOの速度を決めます．指定できる範囲は0～16383までで，大きいほど速くなります.

V%(0, 5): Pitch Modulation Depth(ピッチモジュレーションデプス)

音程に対してLFOをかける深さを決めます．指定できる範囲は−127～127までで，絶対値が大きいほど深くかかります．負の値をとると，波形の極性が逆になります.

V%(0, 6): Amplitude Modulation Depth(アンプリチュードモジュレーションデプス)

音量に対してLFOをかける深さを決めます．指定できる範囲は−127～127までで，絶対値が大きいほど深くかかります．負の値をとると，波形の極性が逆になります.

V%(0, 7): Pitch Modulation Sensitivity(ピッチモジュレーションセンシティビティ)

音程に対するLFOをかける度合いを決定します．指定できる範囲は0～15までで，数値が大きいほど変化の幅が大きくなります．通常，Pitch Modulation Sensitivityで大まかな調整をしておいて，Pitch Modulation Depthで細かく調整します.

V%(0, 8): 未使用

V%(0, 9): 未使用

OPはオペレータ番号を意味します．1～4の値を指定します.

V%(OP, 0): Attack Rate(アタックレート)

音が発生(Key-on)してからEG(Envelop Generator)の出力レベルが最大になるまでの時間を決めます．指定できる範囲は0～31までで，数値が大きいほど短くなります.

V%(OP, 1): Decay Rate(ディケイレート)

EGの出力レベルが最大になってからSustain Levelに達するまでの時間を決めます．指定できる範囲は0～31までで，数値が大きいほど短くなります.

V%(OP, 2): Sustain Rate(サスティーンレート)

Sustain LevelからEGの出力レベルが0になるまでの時間を決めます．指定できる範囲は0〜31までで，数値が大きいほど短くなります．

V%(OP, 3): Release Rate(リリースレート)

Keyを離してからEGの出力レベルが0になるまでの時間を決めます．指定できる範囲は0〜15までで，数値が大きいほど短くなります．

V%(OP, 4): Sustain Level(サスティーンレベル)

DecayからSustainに移るときのレベル(減衰量)を決めます．指定できる範囲は0〜15までで，数値が大きいほどEGの出力レベルが低くなります．

V%(OP, 5): Output Level(アウトプットレベル)

これは減衰量のため，0が最大出力になります．指定できる範囲は127〜0までです．

V%(OP, 6): Keyboard Rate Scaling Depth(キーボードレートスケーリングデプス)

高音になるほどEGの変化速度を速くする度合いを決めます．指定できる範囲は0〜3までです．

V%(OP, 7): Multiple(マルチプル)

周波数比を決定します．指定できる範囲は0〜15までで，0で½になります．

V%(OP, 8): Detune(デチューン)

音程のずれの度合いを決めます．指定できる範囲は−3〜3までです．

V%(OP, 9): Amplitude Modulation Sensitivity(アンプリチュードモジュレーションセンシティビティ)

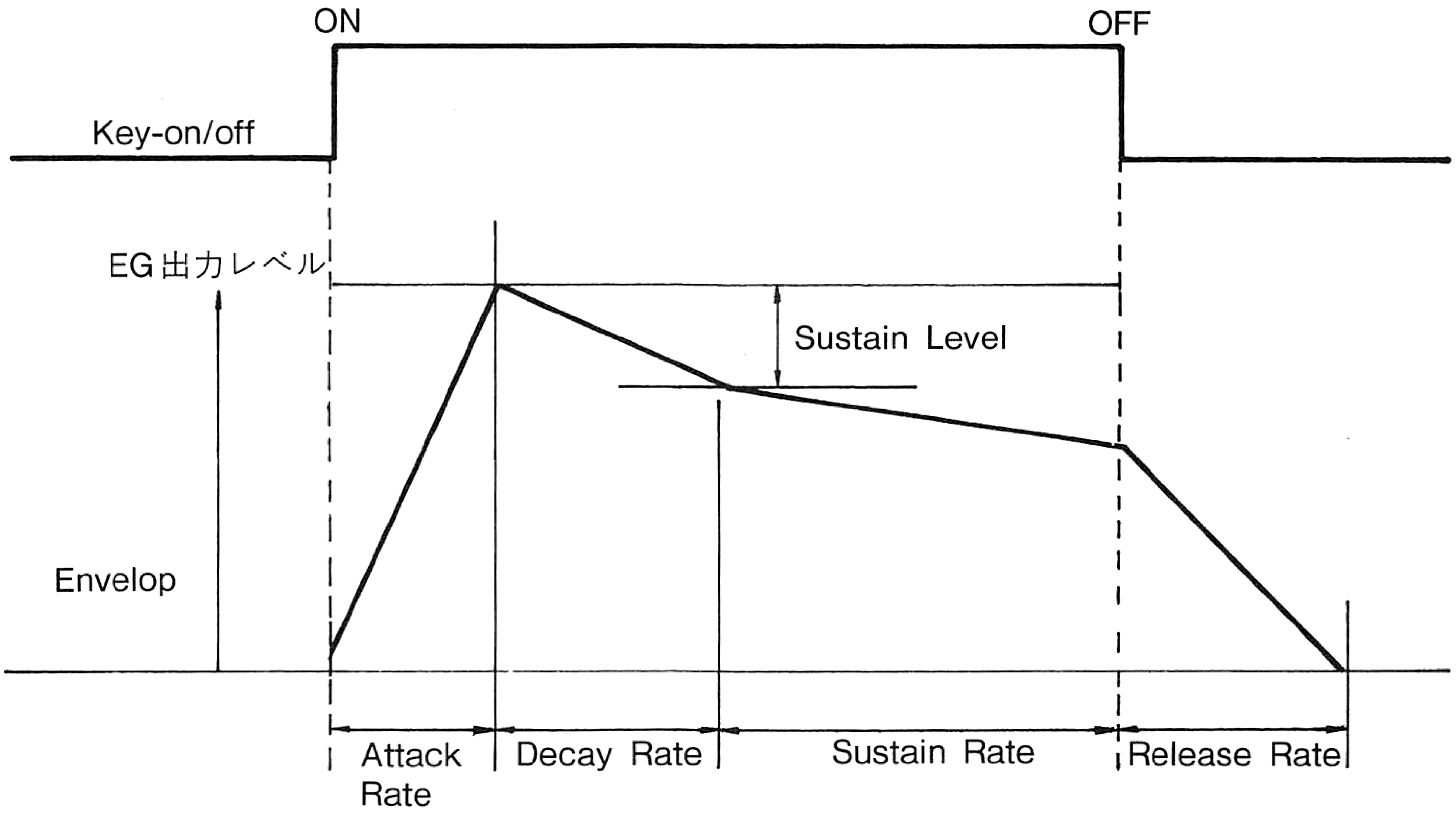
音量に対するLFOをかける度合いを決定します．指定できる範囲は0〜15までで，数値が大きいほど変化の幅が大きくなります．通常，Amplitude Modulation Sensitivityで大まかな調整をしておいて，Amplitude Modulation Depthで細かく調整します．

• EG(Envelop Generator…エンベロープジェネレータ)

楽器の音は時間とともに音量，音色が変化します．この時間的な変化をエン

ベロープといいます．そして，このエンベロープを人工的に作る機能をEG(Envelop Generator)といいます．

EGは以下のパラメータを持っており，それを設定することによって，より自然の楽器に近い表情を付けることができます．



例 VOICE A%, B%, C%

• 整数配列変数A%, B%, C%に入っている音色データを各チャネルに設定します．

プログラム例

```
list
100 ' VOICE sample
110 DIM A%(4,9)
120 BGM 0
130 VOICE COPY 20,A%
140 VOICE A%
150 PLAY "O6V15L1C&C&C&C"
160 FOR W=0 TO 1000:NEXT W
170 A%(0,2)=1
180 A%(0,3)=1
190 A%(0,4)=450
200 A%(0,5)=100
210 A%(0,7)=8
220 VOICE A%
230 PLAY "O6V15L1C&C&C&C"
240 END
Ok
```

• 2種類の音を出します．

• 130行で配列A%に音色番号20(正弦波)のデータをコピーします．

• 140行で，音色番号20のデータをそのままFM音源に送り，150行で音を出します．

- 170行から210行でデータを多少書き換えます.
- 220行で，変更したデータをFM音源に送り，230行で音を出します.

参照：資料4　FM音源とシンセサイザーIC

ミュージックステートメント

VOICE COPY

ボイス・コピー：voice copy

機能 音色データを配列変数にコピーする

書式 VOICE COPY〈音色番号〉,〈整数配列名〉

解説
- 〈音色番号〉で示されたFM音源のパラメータの値を，〈整数配列名〉で指定された配列変数にコピーします．この配列変数は，VOICE文で新しく音色を設定するために用います．
- 配列変数はDIM文で定義し，添字は(4，9)で宣言します．
- 〈音色番号〉についてはPLAY文を参照してください．

例 VOICE COPY 7,F%

- フルートの音色データを整数配列変数F%にコピーします．

プログラム例

```
list
100 ' VOICE COPY sample
110 DIM A%(4,9)
120 BGM 0
130 VOICE COPY 20,A%
140 VOICE A%
150 PLAY "O6V15L1C&C&C&C"
160 FOR W=0 TO 1000:NEXT W
170 A%(0,2)=1
180 A%(0,3)=1
190 A%(0,4)=450
200 A%(0,5)=100
210 A%(0,7)=8
220 VOICE A%
230 PLAY "O6V15L1C&C&C&C"
240 END
Ok
```

- 2種類の音を出します．
- このプログラム例はVOICE文と同じものです．

参照：DIM，PLAY，VOICE

VOICE LFO

ボイス・エル・エフ・オー：voice lfo

機　能　各チャネルにLFO効果を与える

書　式　VOICE LFO〈チャネル番号〉〔,〈wf〉〕〔,〈syncスイッチ〉〕〔,〈スピード〉〕〔,〈pmd〉〕〔,〈amd〉〕〔,〈pms〉〕

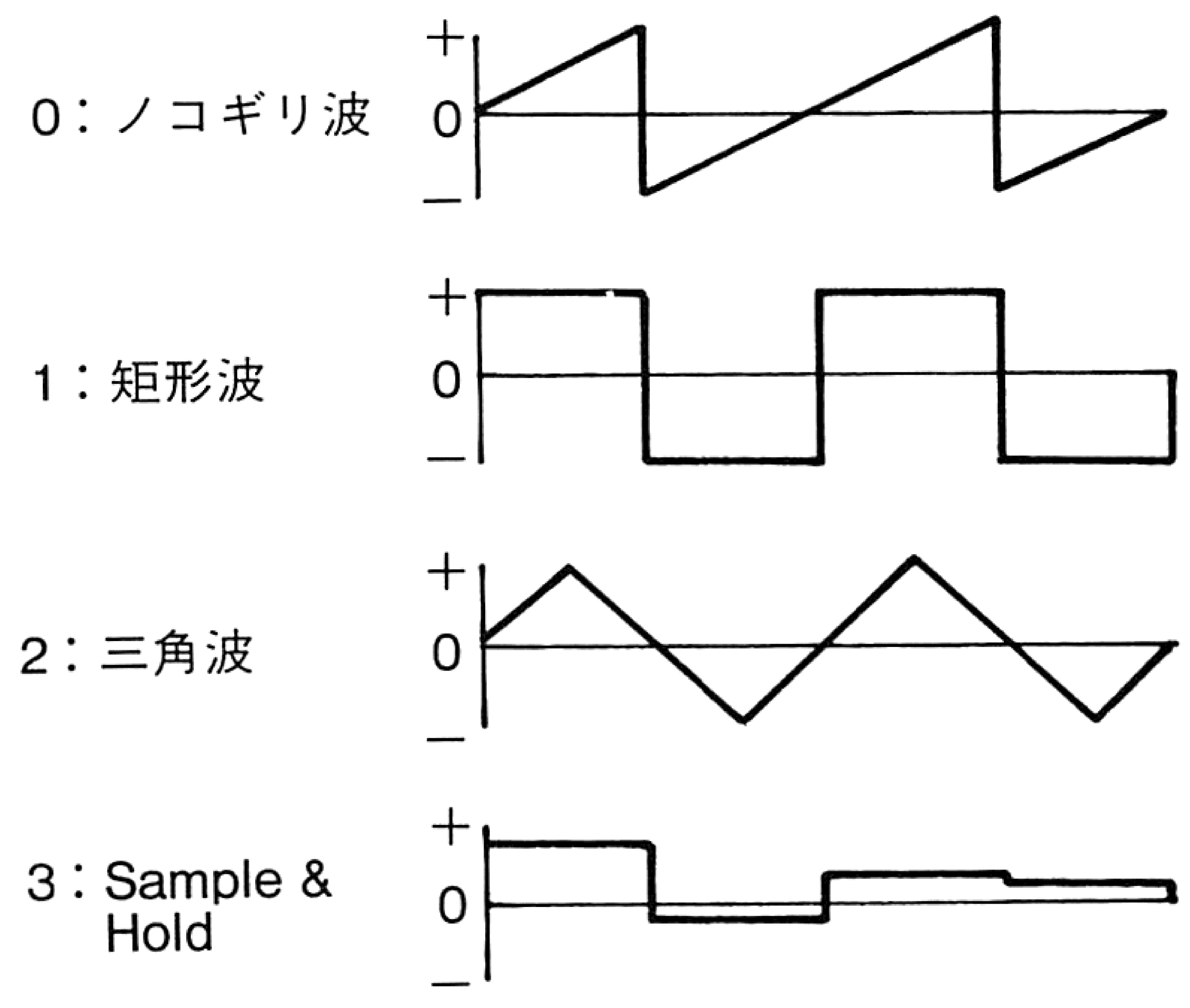
解　説

• 指定したチャネルの音にビブラート(音程を細かく変化させること)やトレモロ(音量を細かく変化させること)を付けます.

• ビブラートやトレモロの効果の度合いは，LOF(Low Frequency Oscillator)と呼ばれる低周波発振器のパラメータによって制御します．パラメータは次のように設定します.

〈チャネル番号〉…1〜6の値を指定します．この値により，LFO効果を付けるチャネルを設定します.

〈wf〉………………Wave Form．波形によりLFOの変化の仕方を設定します．0〜3の値を指定することにより，次のような波形を選ぶことができます.



Sample & Hold(サンプルアンドホールド)はランダム値が出力されます.

〈syncスイッチ〉…synchroスイッチが1の場合，音を発生させると同時にLFO波形を発生させます．0の場合，音の発生と無関係にLFO波形を発生させます.

〈スピード〉………0〜16383の値を指定することにより，LFOの周波数を決定します．数が大きいほど周波数が高くなります.

〈pmd〉……………Pitch Modulation Depth．音程に対してLFOをかける深さを決定します．つまり，ビブラートをかける場合に使います．指定できる値は－127〜127までで，絶対値が大きいほど変化の幅が大きくなります．負の値を指定すると，LFO波形の極性が逆転します．

〈amd〉……………Amplitude Modulation Depth．音量に対してLFOをかける深さを決定します．つまり，トレモロをかける場合に使います．ただし，〈amd〉が設定できるのはFM音源のときだけです．指定できる値は－127〜127までで，絶対値が大きいほど変化の幅が大きくなります．負の値を指定すると，LFO波形の極性が逆転します．

〈pms〉……………Pitch Modulation Sensitivity．〈pmd〉と同様，音程に対してLFOをかける度合いを決定します．指定できる値は0〜15までで，数値が大きくなるほど変化の幅が大きくなります．通常，〈pms〉で大まかな調整をしておいて，〈amd〉で細かい調整をします．

例 VOICE LFO 1,2,1,100,90

• チャネル番号1の音にLFO効果を設定します．

プログラム例

```
list
100 ' VOICE LFO sample
110 BGM 0
120 PLAY "@20V15O6L1"
130 PLAY "C&C&C&C"
140 FOR W=0 TO 1000:NEXT W
150 VOICE LFO 1,3,1,10000,100,,15
160 PLAY "C&C&C&C"
170 END
Ok
```

• 正弦波の音にLFO効果を設定して，ランダムな音を出します．

• 150行で，チャネル1にビブラートをかけるために，wf=3，スピード=10000，pmd=100，pms=15に設定します．

注意：• チャネル1〜6のすべてにLFOをかけると，多少テンポが狂う場合があります．

VOICE REG

ボイス・レジ：voice register

機能 シンセサイザーIC(OPN)のレジスタに値を設定する

書式 VOICE REG〈OPNレジスタ番号〉,〈数式〉

解説
- 指定されたOPNのレジスタに，〈数式〉によって指定された値を代入します．
- 〈数式〉の値の範囲は0～255です．
- OPNレジスタについては，資料4　FM音源とシンセサイザーICを参照してください．

例 VOICE REG &HA0,1

- OPNのレジスタ(A0H)にデータ(1)を設定します．

プログラム例

```
list
100 ' VOICE REG sample
110 STOPM
120 VOICE REG 7,7
130 VOICE REG 6,30
140 VOICE REG 13,12
150 FOR I=1 TO 30
160   FOR L=0 TO 15
170     VOICE REG 8,L
180   NEXT L
190   FOR L=15 TO 0 STEP -3
200     VOICE REG 8,L
210   NEXT L
220   IF I MOD 15=0 THEN PLAY "@53V15O5CC2"
230 NEXT I
240 END
Ok
```

- 蒸気機関車が走っている音を出します．
- 120行でSSG音源の3音がすべてノイズを出力するように設定します．
- 130行と140行で，ノイズの発生周期及びエンベロープタイプを設定します．
- 150行から230行で音量の調節を行います．

WAIT

ウェイト：wait

機能 入力ポートをモニタし，その間プログラムの実行を中断する

書式 WAIT 〈I/Oアドレス〉,〈式1〉〔,〈式2〉〕

解説 • WAIT文は，指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行を中断します．

• 指定したポートから読み込んだデータと〈式2〉とのXORの結果と〈式1〉とのANDが取られ，その結果が真(0以外)になるまでポートのデータを読み続けます．XORとANDの結果をAとして式で表せば次のようになります．

A＝((読み込んだデータ)XOR〈式2〉)AND〈式1〉

この結果(この場合A)が0(偽)であれば，BASICはもう一度ポートのデータを読み込み同じ操作を繰り返します．もし結果が0でなければ(真ならば)，プログラムの実行は次の文に移ります．

• 〈式2〉が省略された場合は0とみなします．

例 WAIT 4,8,255

• I/Oポートの4をモニタし，ビット3が0になるのを待ちます．つまり，"s"キーが押されるのを待ちます．

プログラム例

```
list↵
100 ' WAIT sample
110 PRINT "Type 's' to start."
120 WAIT 4,8,255
130 END
Ok
run↵
Type 's' to start.
Ok
s
```

• "s" キーを押すまで待ち続けます．

• 120行で，"s" のキーが押されるまで待ち続けます．

注意：• WAIT文は，式の指定を誤ると無限ループに入ってしまう場合があります．その場合にはPC-8001MKⅡSRをリセットしなければなりません．

WIDTH

ウィドゥス：width

機能 テキスト画面に表示する文字の行数と桁数を指定する

書式 WIDTH〔〈桁数〉〕〔,〈行数〉〕

解説
- テキスト画面に表示する1画面当たりの〈行数〉と1行当たりの〈桁数〉を指定します.
- 〈桁数〉は80か40のどちらかを指定します. ただし, テキスト画面がカラーモードの状態にあるときは, 1行当たりに表示する桁数は〈桁数〉−1となります(白黒モードの場合, 表示桁数＝〈桁数〉になります). また, 〈桁数〉が省略された場合, 現在の〈桁数〉が採用されます.
- 〈行数〉は20か25のどちらかを指定します. 省略された場合, 現在の〈行数〉が採用されます.
- WIDTH文を実行すると, テキスト画面は消去されます.

例 WIDTH 80,25
- テキスト画面を80桁, 25行に設定します.

プログラム例

```
list↵
100 ' WIDTH sample
110 WIDTH 80,20:CLS
120 GOSUB 170
130 IF INKEY$="" THEN 130
140 WIDTH 80,25:CLS
150 GOSUB 170
160 END
170 PRINT "┌───┬───┬───┬───┐"
180 PRINT "├───┼───┼───┼───┤"
190 PRINT "└───┴───┴───┴───┘"
200 RETURN
Ok
```

- 初めにテキスト画面を80桁×20行に設定して罫線を描き, 次に80桁×25行で描きます.
- 110行から120行で, 80桁×20行で罫線を描きます. この場合, 線が接続されない部分があります.
- 130行は, 80桁×20行で罫線を描いたあと, 次の罫線を描くのを待つためにあります. 任意のキーを押すことによって次の処理に移ります.
- 140行から150行で, 80桁×25行で罫線を描きます. このときは線がすべて接続されます.

参照：CONSOLE

第3章

N80SR-BASICの応用

3.1　グラフィック命令を使ったプログラム

N_{80SR}-BASICはCIRCLE文，PAINT文，GET@文，PUT@文などの様々なグラフィック命令を持っています．さらにN_{80SR}-BASICは4種類の画面モードを持ち，画面の優先順位を付けることもできます．これらを組み合わせることによってN_{80SR}-BASICは多彩なグラフィックを表示することが可能です．

ここでは，いろいろなグラフィック命令を組み合わせたプログラムを紹介します．

3.1.1　白黒モードでのグラフィックの使い方

N_{80SR}-BASICの白黒モードでは，640×200ドットの画面を3ページ，又は，320×200ドットの画面を6ページ使用することができ，任意のページに書き込んだり表示することができます．したがって，表ページを見せながら裏ページに書き込んだり，既に各ページに書き込んである図形を重ね合わせたりして，より効果的な画面を作ることができます．

ここでは，320×200ドットの白黒モードの各ページにそれぞれ比率の異なる楕円を描き，描き終わった時点で1ページずつ順に表示するプログラムを紹介します．

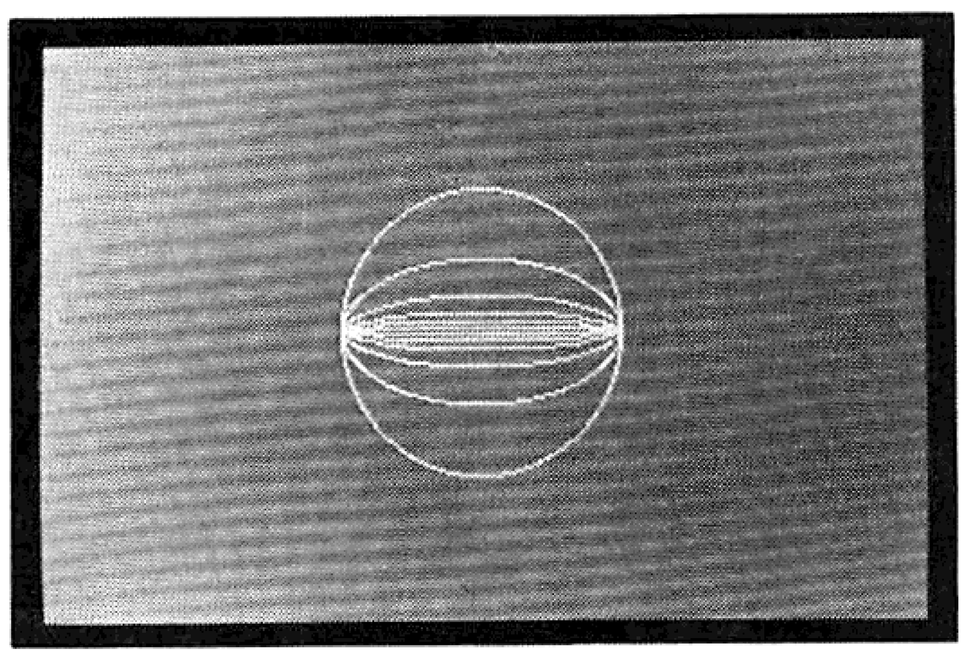
- 130行から140行までは，楕円を描くために画面モードの設定や画面の消去を行っています．
- 150行から170行までは，ディスプレイページ及びアクティブページを切り替えながら，比率の異なる楕円を描いています．
- 180行から260行までは，実際の画面切り替えで，ここでディスプレイページを高速で切り替えることによって楕円が振動しているように見せています．
- 270行では，画面の切り替え後に"PC-8001mk2SR"と表示しています．

プログラム中では2つの変数が使われており，使用目的は次のとおりです．

使用した変数名	変数の使用目的
I	ページ番号を増加・減少させるため
J	画面の切り替え回数を数えるため

```
100 ' 320*200ﾄﾞｯﾄ ｼﾛｸﾛﾓｰﾄﾞ ｸﾞﾗﾌｨｯｸ !
110 '                          ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
120 '
130 WIDTH 40,25:CONSOLE 0,25,0,1
140 COLOR 7,1,0,7:SCREEN 0,0:CLS 3
150 FOR I=0 TO 5
160  SCREEN 3,0,I,2^I:CIRCLE(159,99),50,,,,2^-I
170 NEXT I
180 FOR J=1 TO 20
190 PLAY"@18V15B"
200  FOR I=0 TO 5
```

```
210   SCREEN ,,,2^I
220   NEXT I
230   FOR I=4 TO 1 STEP -1
240    SCREEN ,,,2^I
250   NEXT I
260  NEXT J
270  LOCATE 13,12:PRINT"PC-8001mk2SR"
280  END
```



3.1.2　320×200ドットカラーモードでの表示優先順位機能の使い方

N80SR-BASICのカラーモードでは，320×200ドットの画面を2ページ使用することができます．又，グラフィック画面の各ページに対して表示優先順位を付けたり，テキスト画面とグラフィック画面の間に表示優先順位を付けることができます．

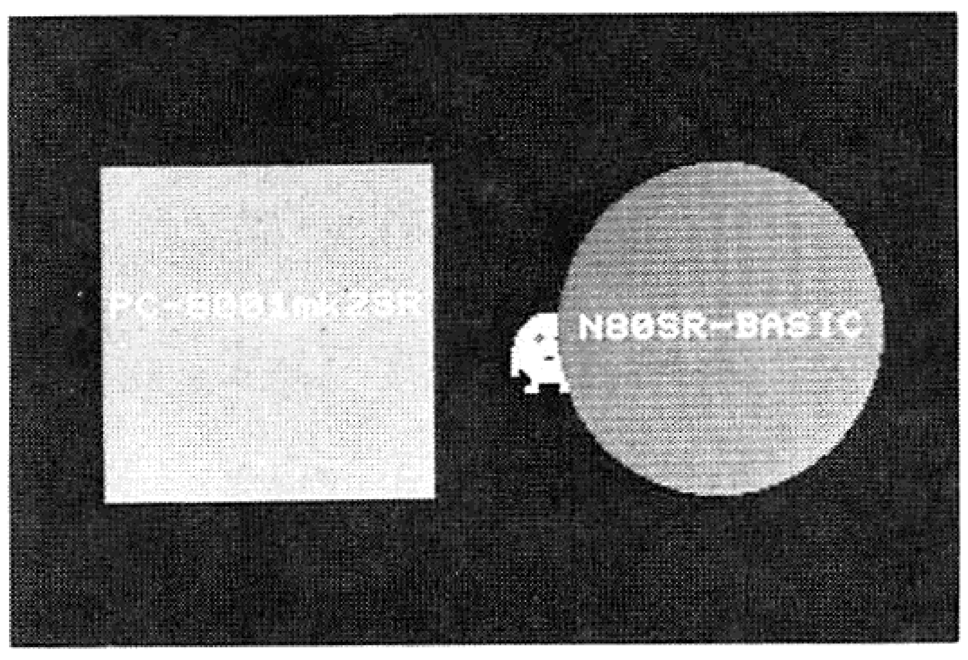
ここでは，0ページ目に正方形と円を描き，1ページ目に表示させた絵を移動させながら画面の表示優先順位を切り替えるプログラム例を示します．

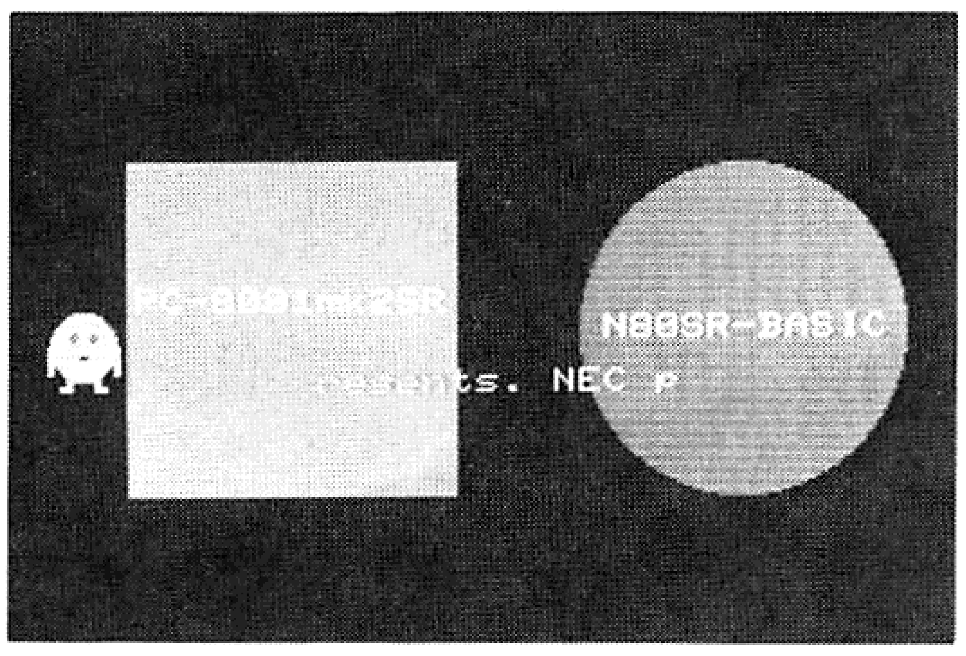
- 130行から170行までは，画面モードの設定及び画面の消去を行い，1ページ目で移動させる絵を格納するための数値配列を確保した後，画面に"PC-8001mk2SR"，"N80SR-BASIC"と表示します．
- 180行から200行までは，第0ページ優先表示・第0ページ書き込みモード・全画面表示モードに設定し，(30, 50)と(130, 150)の2点を対角とする正方形を描き，次に(219, 100)を中心として半径50の円を描きます．
- 210行から310行までは，第0ページ優先表示・第1ページ書き込みモード・全画面表示モードに設定し絵を描き，GET@文を用いて(84, 85)から(117, 118)までの領域を数値配列に読み込みます．
- 320行では，第1ページのみを消去しています．
- 330，340行では第1ページを表示優先にして，第1ページ上で図形を移動させています．
- 350，360行では第0ページを表示優先にして，第1ページ上で図形を移動させています．
- 370，380行では第1ページを表示優先にして，第1ページ上で図形を移動させています．
- 390，400行では第0ページを表示優先にして，第1ページ上で図形を移動させています．
- 410行から460行では，テキスト画面とグラフィック画面の表示優先順位を切り替えながら，画面に"NEC presents."と表示します．
- 470行から520行では，再度グラフィック画面の優先順位を切り替えながら第1ページ上で図形を移動させます．

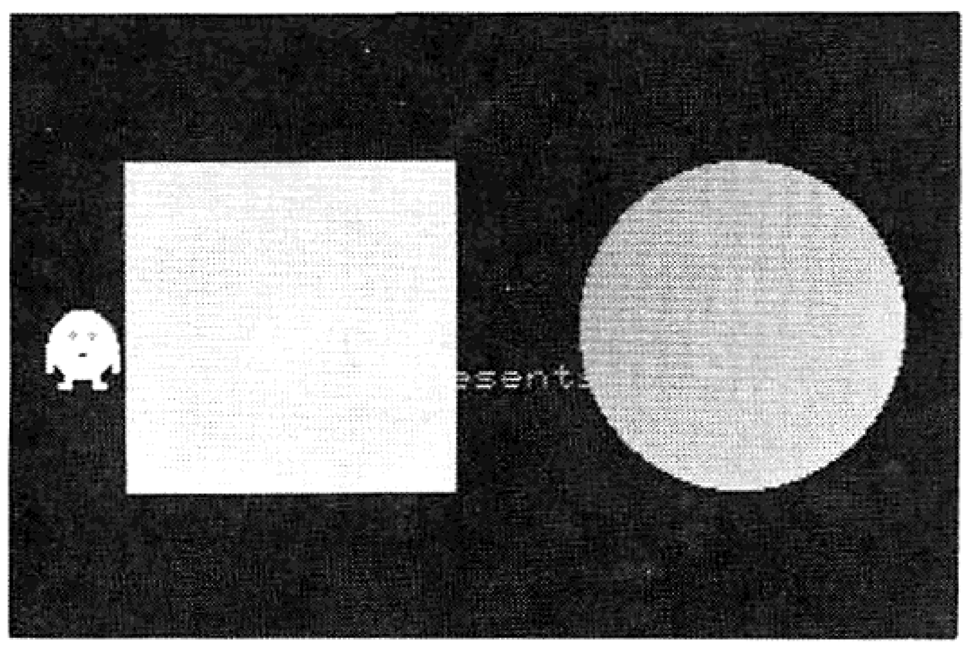
このプログラムでは変数が4つ使われており，その使用目的は次のようになります．

使用した変数名	変数の使用目的
G%	絵を読み込むための整数型配列(要素数257個)
I	X座標を順次変化させるため
J	"NEC presents. "を表示する回数を数えるため
K	何番目の文字から表示するかを数えるため

```
100 ' 320*200ﾄﾞｯﾄ ｶﾗｰﾓｰﾄﾞ ｸﾞﾗﾌｨｯｸ/ﾃｷｽﾄ
110 '                ﾕｳｾﾝｼﾞｭﾝｲ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
120 '
130 CONSOLE 0,25,0,1:COLOR 7,0,0,7:SCREEN 0
140 WIDTH 40,25:CLS 3
150 DIM G%(256)
160 LOCATE 3,11:PRINT"PC-8001mk2SR"
170 LOCATE 21,12:PRINT"N80SR-BASIC"
180 SCREEN 2,0,0,3
190 LINE(30,50)-(130,150),1,BF
200 CIRCLE(219,100),50,2:PAINT(219,100),2,2
210 SCREEN 2,0,1,3
220 LINE(95,108)-(96,112),3,BF
230 LINE-(93,113),3,BF
240 LINE(105,108)-(104,112),3,BF
250 LINE-(107,113),3,BF
260 LINE(89,100)-(90,108),3,BF
270 LINE(111,100)-(110,108),3,BF
280 CIRCLE(100,100),10,6:PAINT(100,100),6,6
290 CIRCLE(97,97),1,0:CIRCLE(103,97),1,0
300 LINE(99,103)-(101,103),0
310 GET@(84,85)-(117,118),G%
320 CLS 2
330 SCREEN ,2:FOR I=0 TO 140 STEP 5
340 PUT@(I,90),G%,PSET:NEXT I
350 SCREEN ,0:FOR I=145 TO 280 STEP 5
360 PUT@(I,90),G%,PSET:NEXT I
370 SCREEN ,2:FOR I=275 TO 145 STEP -5
380 PUT@(I,90),G%,PSET:NEXT I
390 SCREEN ,0:FOR I=140 TO 0 STEP -5
400 PUT@(I,90),G%,PSET:NEXT I
410 A$="NEC presents. ":A$=A$+A$
420 FOR J=1 TO 10:PLAY"@16O4T160L4C"
430 FOR K=1 TO 14:LOCATE 10,14
440 COLOR (K-1) MOD 7+1
450 PRINT MID$(A$,K,14):NEXT K
460 SCREEN ,2+4*(J MOD 2):NEXT J
470 SCREEN ,0:FOR I=0 TO 90 STEP 5
480 PUT@(I,90-I),G%,PSET:NEXT I
490 SCREEN ,2:FOR I=95 TO 255 STEP 5
500 PUT@(I,I-90),G%,PSET:NEXT I
510 LOCATE 30,20:PRINT"Thanks"
520 END
```







3.1.3 タイリングペイントの使い方

タイリングは"タイル"の大きさとパターンを適度に設定することによって，服地の模様のシミュレーションや中間色の表現など広範囲な応用が考えられます．

次のプログラムは，2つの点を組み合わせての27色表現と，乱数による37パターンのタイルを表示するものです．このプログラムを実行する場合は，N₈₀SR-BASIC DISKバージョンを起動してください．

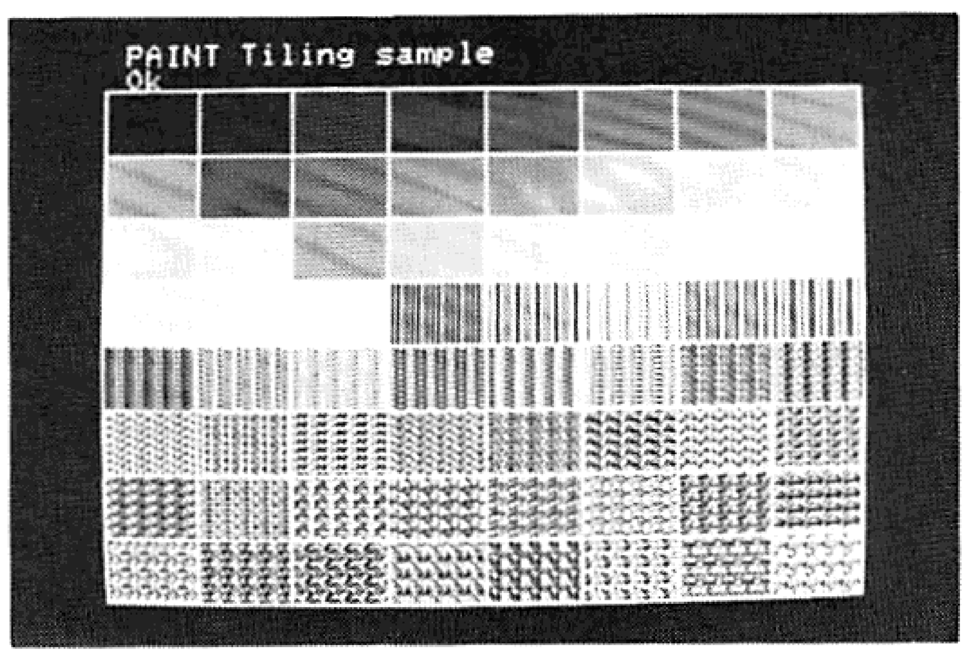
- 120行から150行までは初期設定で，画面のモードの設定等を行っています．
- 160行から190行までは，27色の基本となるパターンを設定しています．このときBASICは自動的に10個の配列を確保してくれます．
- 200行から220行までは，縦線を9本描くためのものです．
- 230行から250行までは，横線を9本描くためのものです．これで，64個の長方形を描きます．
- 260行から320行までは，27色のタイリングペイントをするもので，300行でタイリングパターンを作成し，310行でペイントしています．
- 330行から380行までは，乱数でタイリングパターンを作成しペイントしています．タイリングパターンの長さは0.5ずつ増加させています．ただし，カラーモードでPAINT文を実行した場合，ストリング中の文字数を3で割った余りの文字は無視されます．たとえば，8文字からなるタイルストリングがあった場合，7文字目と8文字目は無視されます．

このプログラムの中で使われている変数とその使用目的は次のとおりです．

使用した変数名	変数の使用目的
A$, B$	タイルの基本パターンを格納する文字型配列
X	縦線を引くためのX座標
Y	横線を引くためのY座標
G, R, B	タイル基本パターンの配列要素番号
P	ペイントする箱の番号
T	タイルストリングの長さを数えるため
TI$	ペイントするタイルストリング

```
100 ' ﾀｲﾘﾝｸﾞ ﾍﾟｲﾝﾄ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 CLEAR 1000
130 CONSOLE 0,25,0,1:COLOR 7,0,0,7
140 SCREEN 2,0,0,1
150 CLS 3:PRINT"PAINT Tiling sample"
160 A$(0)=CHR$(0):A$(1)=CHR$(&H55)
170 A$(2)=CHR$(&HFF)
180 B$(0)=CHR$(0):B$(1)=CHR$(&HAA)
```

```
190 B$(2)=CHR$(&HFF)
200 FOR X=0 TO 312 STEP 39
210   LINE(X,15)-(X,199),7
220 NEXT X
230 FOR Y=15 TO 199 STEP 23
240   LINE(0,Y)-(312,Y),7
250 NEXT Y
260 FOR G=0 TO 2
270   FOR R=0 TO 2
280     FOR B=0 TO 2
290       P=G*9+R*3+B
300       TI$=A$(B)+A$(R)+A$(G)+B$(B)+B$(R)+B$(G)
310       PAINT((P MOD 8)*39+1,(P ¥ 8)*23+16),TI$,7
320 NEXT B,R,G
330 FOR P=27 TO 63
340   TI$=""
350   FOR T=1 TO (P-26)/2+3
360     TI$=TI$+CHR$(RND(1)*256):NEXT T
370   PAINT((P MOD 8)*39+1,(P ¥ 8)*23+16),TI$,7
380 NEXT P
390 END
```



3.1.4 パレットの使い方

N80SR-BASICのパレット機能をうまく使用することにより，画面上で静止している図形をあたかも動いているかのように見せることもできます．

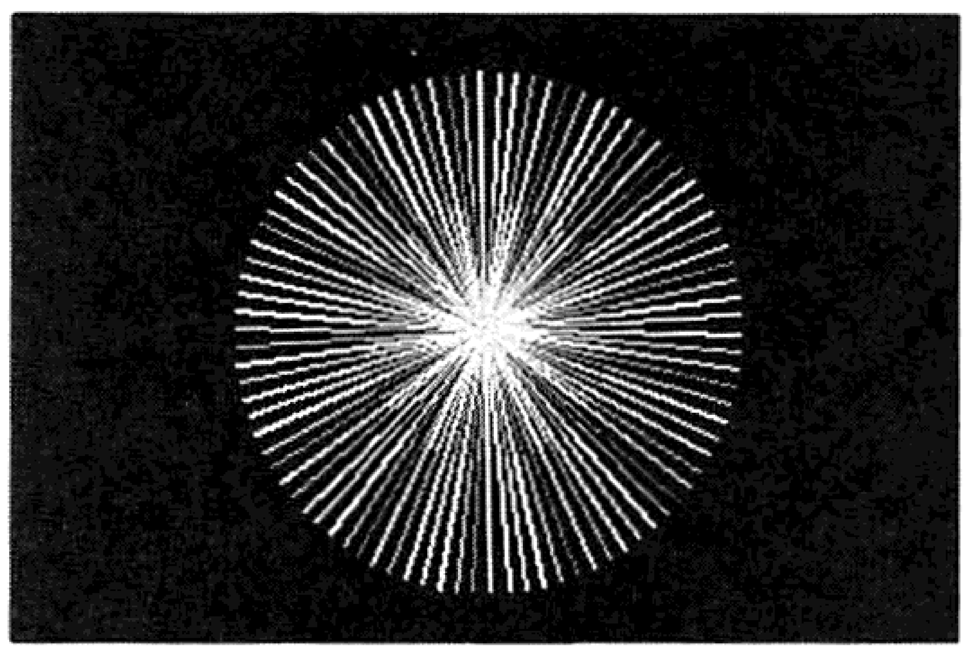
ここでは，画面に放射線を描いてパレット１から７までを回転させるプログラムを紹介します．

- 120行から130行までは画面の初期設定で，画面のモードを設定し画面を消去しています．
- 140行，150行は，カラーパレット０を０(黒)に，その他のパレットを７(白)に設定しています．
- 160行から230行では，第０ページを用いて半径45までの放射線と半径45から半径90までの放射線をそれぞれカラーパレットを増加，減少させながら描きます．
- 240行から300行では，カラーパレットを順次回転させています．カラーパレット１から７までの内容を１ずつ増加(内容が８となった場合は，１に戻す)させることにより回転させることができます．
- 310行では，カラーパレットをBASICの初期値に戻しています．カラーパレットを使用した後に，カラーパレットをBASICの初期値に戻さないと，他のプログラムを実行するときに色がおかしくなったり，グラフィックが見えなくなる場合があるので注意が必要です．

このプログラムで使用されている変数とその使用目的は次のとおりです．

使用した変数名	変数の使用目的
I	パレット番号
C1	半径45の放射線のカラーパレット番号
C2	半径45から90までの放射線のカラーパレット番号
DE	放射線の角度(度)
RA	放射線の角度(ラジアン)
OF	パレットを回転させるときのパレットの基準値
K	パレットを回転させる回数を数えるため

```
100 ' ﾊﾟﾚｯﾄ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 CONSOLE 0,25,0,1:COLOR 7,0,0,7
130 SCREEN 2,0,0,1:CLS 3
140 COLOR=(0,0)
150 FOR I=1 TO 7:COLOR=(I,7):NEXT
160 C1=1:C2=4
170 FOR DE=0 TO 359 STEP 4
180   RA=3.14159/180*DE
190   LINE(159,99)-(159+45*SIN(RA),99-45*COS(RA)),C1
200   LINE-(159+90*SIN(RA),99-90*COS(RA)),C2
210   C1=C1+1:IF C1=8 THEN C1=1
220   C2=C2-1:IF C2=0 THEN C2=7
230 NEXT DE
240 OF=1
250 FOR K=1 TO 100
260 FOR I=1 TO 7
270   COLOR=(I,(OF+I) MOD 7+1)
280 NEXT I
290 OF=OF+1:IF OF=6 THEN OF=0
300 NEXT K
310 FOR I=0 TO 7:COLOR=(I,I):NEXT
320 END
```



3.2　サウンド機能を使ったプログラム

PC-8001MKⅡSRにはシンセサイザーIC(OPN)が内蔵されており，そのサウンド機能を使用することによって，さまざまな音色を出したりあるいは重音演奏することができます．

ここでは，サウンド機能を使ったプログラミングの方法を紹介します．

3.2.1　単音による演奏

曲の演奏はMML(Music Macro Language)を使って，PLAY文で行います．

次のプログラム例はチャネル1のFM音源で単音による演奏を行います．

• 110行でSTOPM文を実行し，ボリューム，オクターブ等を初期化しています．この命令を実行しなかった場合，以前に設定されたパラメータの影響を受けることがありますのでプログラムの最初で必ず実行するようにします．

• 120行では音色をハープシコード，オクターブを5，テンポを170，音符の長さを8(8分音符)に設定しており，それ以外の機能(ボリュームや音の長さの割合等)は初期値が設定されています．

• 130行から200行では実際に出力する音を指定して，曲の演奏をしています．

```
100 ' ﾀﾝｵﾝ ﾃﾞ ｴﾝｿｳ ｽﾙ
110 STOPM
120 PLAY "@0O5T170L8"
130 PLAY "G4CDEFG4Q3C4C4"
140 PLAY "Q8A4FGAB>C4<Q3C4C4"
150 PLAY "Q8F<F>GFEDE<E>FEDC<"
160 PLAY "B<B>>CDECE4Q3D4<G4>"
170 PLAY "Q8G4CDEFG4Q3C4C4"
180 PLAY "Q8A4FGAB>C4<Q3C4C4"
190 PLAY "Q8F<F>GFEDE<E>FEDC"
200 PLAY "D<D>EDC<B>C3"
210 END
```

3.2.2　音色を変えて演奏する

内蔵されているOPNには，いろいろな楽器の音が用意されています．これらの楽器の音は音色番号0から61までに定義されており，PLAY文の@コマンドで音色番号を指定することによって演奏できます．次の表は音色番号表です．

音色番号表

音色番号：音色名	音色番号：音色名
0：Default VOICE	31：EBASS 2
1：BRASS 2	32：EORGAN 1
2：STRING 2	33：EORGAN 2
3：EPIANO 3	34：PORGAN 1
4：EBASS 1	35：PORGAN 2
5：EORGAN 1	36：FLUTE
6：PORGAN 1	37：PICCOLO
7：FLUTE	38：OBOE
8：OBOE	39：CLARINET
9：CLARINET	40：GROCKEN
10：VIBRPHN	41：VIBRPHN
11：HARPSIC	42：XYLOPHN
12：BELL	43：KOTO
13：PIANO	44：ZITAR
14：MUSHI	45：CLAV
15：DESCENT	46：HARPSIC
16：UFO	47：BELL
17：GRANPRI	48：HARP
18：LASER 1	49：BELL/BRASS
19：LASER 2	50：HARMONICA
20：SIN WAVE	51：STEEL DRUM
21：BRASS 1	52：TIMPANI
22：BRASS 2	53：TRAIN
23：TRUMPET	54：AMBULAN
24：STRING 1	55：TWEET
25：STRING 2	56：RAIN DROP
26：EPIANO 1	57：HORN
27：EPIANO 2	58：SNARE DRUM
28：EPIANO 3	59：COW BELL
29：GUITAR	60：PERC 1
30：EBASS 1	61：PERC 2

PLAY文中に@コマンドが指定されていない場合，以前に音色番号が指定されていたときはその音色で，そうでない場合はDefault VOICEのハープシコードで演奏されます．

次のプログラムは上記の音色番号を利用していろいろな音で単音による演奏を行うものです．

• 120行で"ネイロ　バンゴウ　ヲ　ニュウリョク　シテクダサイ。"というメッセージを表示し，音色番号の入力を促します．

• 0未満や61より上の数字を指定した場合は，130行でビープ音を出し120行に戻ります．

• 150行では，READ文で読み込むDATA文の行番号を220行に指定します．

• 160行で，音楽機能の初期化を行い，並列動作モードをOFFにします．並列動作モードをOFFにすることによって，演奏している間は次の命令に移るのを待ちますから，演奏し終ってから120行のメッセージを表示することができます．

• 180行のPLAY文で，変数Nに読み込んだ音色番号を定義します．

• 190行のREAD文で，"end"が読み込まれるまで音符データを読み込み，200行で演奏します．

プログラムを終了したい場合は STOP キーを押します．

```
100 ' ﾈｲﾛ ｦ ｶｴﾃ ｴﾝｿｳ ｽﾙ
110 CLS
120 INPUT "ﾈｲﾛ ﾊﾞﾝｺﾞｳ ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ｡";N
130 IF N<0 OR N>61 THEN BEEP:GOTO 120
140 '
150 RESTORE 220
160 STOPM:BGM 0
170 PRINT "ｴﾝｿｳﾁｭｳ..."
180 PLAY "@=n;"
190 READ MM$:IF MM$="end" THEN 120
200 PLAY MM$:GOTO 190
210 '
220 DATA "O5T170"
230 DATA "L8G4CDEFG4Q3C4C4"
240 DATA "Q8A4FGAB>C4<Q3C4C4"
250 DATA "Q8F<F>GFEDE<E>FEDC<"
260 DATA "B<B>>CDECE4Q3D4<G4>"
270 DATA "Q8G4CDEFG4Q3C4C4"
280 DATA "Q8A4FGAB>C4<Q3C4C4"
290 DATA "Q8F<F>GFEDE<E>FEDC"
300 DATA "D<D>EDC<B>C3"
310 DATA "end"
320 END
```

3.2.3　効果音の使い方

OPNに用意されている音色には楽器の音の他に，虫の鳴き声やレーザーガンなどの効果音もあります．これらの効果音はPLAY文を使って，曲を演奏するときのプログラムを多少変えるだけで出すことができます．プログラム例を3つほどあげて，説明していきます．

(1)　汽車の音を鳴らすには

音色番号53を使って汽車の音を鳴らします．この53番の音はオクターブを低く設定することによってノイズを強調することができます．

• 120行から140行で，ノイズを利用してテンポをだんだん速くしていくことにより汽車が

走り出す蒸気音を作ります.

- 150行では，オクターブを高く指定して警笛を鳴らします.
- 160行から180行で，再びオクターブを低く設定し汽車の蒸気音を出します.

```
100 ' コウカオン 1 ---  キシャ ノ オト
110 STOPM
120 FOR TP=50 TO 255 STEP 15
130   PLAY "@53T=TP;V8O4Q4CV5CCC"
140 NEXT
150 PLAY "V8O7Q8C8R4C1"
160 FOR C=1 TO 20
170   PLAY "T213V8O4Q4C8.V5C8.C8.C8."
180 NEXT
190 END
```

(2) 救急車のサイレンの音を鳴らすには

音色番号54を利用して救急車のサイレンの音を鳴らします.

次のプログラムではチャネル1，チャネル2のFM音源を利用して音作りをしています.

- 120行から140行で，ボリュームを次第に大きくすることによって，サイレンの音が近づいてくる様子を表しています.
- 150行から170行では，逆にサイレンの音が遠ざかる様子を表すため，ボリュームを小さくしていきます.
- ドップラー効果を[注*]表すため160行で指定した音程は，130行の音程よりも半音下げてあります.

```
100 ' コウカオン 2 --- キュウキュウシャ ノ オト
110 STOPM
120 FOR VM=0 TO 14 STEP 2
130   PLAY "@54V=VM;O5B2","@54V=VM;O5A#2"
140 NEXT
150 FOR VM=14 TO 0 STEP -2
160   PLAY "V=VM;A#2","V=VM;A2"
170 NEXT
180 END
```

(3) ベル&ブラス

音色番号49はこれ1つでベルの音とブラスの音を出すことができます. スタッカートにするとベルの音に，1音を長くのばすとブラスの音になります.

次のプログラムは音色番号49を使ったプログラムです.

注＊　ドップラー効果

救急車は同じ音を鳴らしながら走っているのですが，近づいてくるときは高い音に，遠ざかるときには低い音に聞こえます. この現象をドップラー効果といいます.

• 120行と150行でオクターブ4，音の長さ4（4分音符)に設定したブラスの音で演奏します.

• 130行と140行でオクターブ7，音の長さ64(64分音符)に設定したベルの音で演奏を行います.

```
 100 ' コウカオン 3 --- ベル ト ブラス ノ オト
110 STOPM
120 PLAY "@4904L4CCGGAAG2FFEEDDC2"
130 PLAY "T12807L64GR4GR4FR4FR4ER4ER4DR2"
140 PLAY "GR4GR4FR4FR4ER4ER4DR2"
150 PLAY "T12504L4CCGGAAG2FFEEDDC2"
160 END
```

3.2.4 演奏しながら絵を描く

音楽を演奏しながら画面に絵を描く場合は，BGM文を使用します.

このBGM文は並列動作モードのON, OFFを行います．並列動作モードをONにすると，演奏している間に次の命令に移ることができますから，演奏しながら絵を描くといったことが可能です．並列動作モードをOFFにした場合，演奏している間は次の命令に移りませんから，「3.2.2　音色を変えて演奏する」のプログラム例のように，演奏が終わってからメッセージを表示することができます.

次のプログラムは並列動作モードをONにして，演奏しながらいろいろな色で四角形を描いたり，消したりするものです.

• 110行から130行までは初期設定で，変数を整数型に，画面をカラーモードⅡにしてからテキスト画面とグラフィック画面の両方を消去しています.

• 150行では，音楽機能を初期化し，並列動作モードをONにしています.

• 160行で，"end"を読み込むまで音符データを読み込みます.

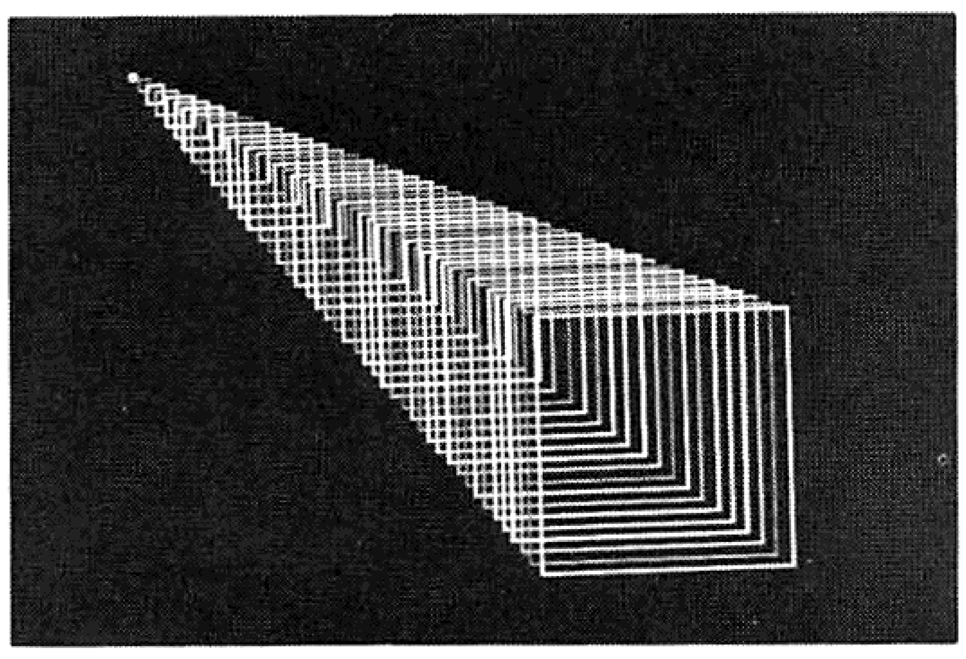
• 170行では，演奏を行います．このとき，並列動作モードがONの状態ですから演奏しながら絵を描くことができます.

• 180行で，絵を描くサブルーチンを呼び，画面上に絵を描きます.

• 330行から400行がそのサブルーチンで，330行から360行で座標を右下にずらしながら徐々に大きな四角形を描き，370行から390行では逆に座標を左上にずらしながら四角形を消していきます.

```
100 ' オト ヲ ダシナガラ エ ヲ カク
110 DEFINT A-Z
120 SCREEN 2,0,0,1
130 CLS 3
140 '
150 STOPM:BGM 1
160 READ MM$:IF MM$="end" THEN END
170 PLAY MM$
```

```
180 GOSUB 320
190 GOTO 160
200 '
210 DATA "O5T170@0"
220 DATA "L8G4CDEFG4Q3C4C4"
230 DATA "Q8A4FGAB>C4<Q3C4C4"
240 DATA "Q8F<F>GFEDE<E>FEDC<"
250 DATA "B<B>>CDECE4Q3D4<G4>"
260 DATA "Q8G4CDEFG4Q3C4C4"
270 DATA "Q8A4FGAB>C4<Q3C4C4"
280 DATA "Q8F<F>GFEDE<E>FEDC"
290 DATA "D<D>EDC<B>C3"
300 DATA "end"
310 END
320 '
330 FOR I=2 TO 90 STEP 2
340  C=((I¥2)MOD 7)+1
350  LINE(I*1.6,I)-STEP(I,I),C,B
360 NEXT
370 FOR I=90 TO 2 STEP -2
380  LINE(I*1.6,I)-STEP(I,I),0,B
390 NEXT
400 RETURN
```



3.2.5 VOICE COPY 文を使って音色を作る

OPNには49種類もの音色が用意されていますが，さらにユーザが新しく音色を作ることも可能です．音色を作るには，すでにOPNに用意されている音色を変えていくことにより新しく音を作る方法と，全く何もないところから新しい音色を作っていく方法の2種類があります．

ここでは，VOICE COPY 文を使って音色番号37のピッコロの音から口笛の音を作るプログラムを紹介します．

口笛の音はピッコロに比べると

1. 音の立ち上りが遅い
2. 音が細い
3. ビブラート（音程を細かく変化させる）が強くかかっている

等の特徴があげられます．音色番号37の以上の点を変更し，演奏を行うのが次のプログラムです．

- 130行で，音色データを格納する配列変数V％を宣言します．
- 140行で，音色番号37のピッコロのデータをV％にコピーします．
- 細い音を出すために，150行でアルゴリズム7を指定し，160行でオペレータ1の使用を禁止します．
- 170行から190行では，音の立ち上りを遅くするためオペレータ2，3，4のAttack Rateを10に設定します．
- 200行でビブラートのかかる度合いを7に，210行でビブラートのスピードを6000に指定し

ます.

• 220行で配列変数V%の口笛のデータをチャネル1のFM音源に送り，240行から280行で単音による演奏を行います.

```
100 ' ネイロ ヲ ツクル
110 '
120 STOPM
130 DIM V%(4,9)
140 VOICE COPY 37,V%
150 V%(0,0)=7
160 V%(0,1)=14
170 FOR OP=2 TO 4
180 V%(OP,0)=10
190 NEXT
200 V%(0,7)=7
210 V%(0,4)=6000
220 VOICE V%
230 '
240 PLAY "O5"
250 PLAY "E2D8C8E8D8C>C<A8>C4&C16<R16G2ECD2.R4"
260 PLAY "E2D8C8E8D8C>C<A8>C4&C16<R16GE8C8DDC2.R4"
270 PLAY "B4.>C8D<GG4.A8G8.R16>CC<AFAG2.R4"
280 PLAY "E2D8C8E8D8C>C<A8>C4&C16<R16GE8C8DDC2.R4"
290 END
```

音色番号で指定できる音色からでなく，全くなにもないところから音色を作る場合は,資料4 FM音源とシンセサイザーICを参照してください.

3.2.6 重音による演奏

重音演奏を行うプログラムを作成する際に重要なことは，プログラムを見やすく作るということです．PLAY文によるプログラムはほとんどMMLのかたまりですから，プログラムが見づらいと修正や変更が大変です.

次のプログラムはREAD文を使ってDATA文に定義されている音符データを読み出し，PLAY文で演奏するものです．ここでは旋律部分のデータ(行番号が偶数のDATA文)と伴奏部分のデータ(行番号が奇数のDATA文)とに分けて入力されていますから，プログラムが見やすく修正や変更が簡単に行えます.

• 130行で音色をピアノに設定します.

• 140行のREAD文で，変数M1$に旋律部分のデータを，変数M2$に伴奏部分のデータを読み込みます．変数M1$に"end"が読み込まれたらプログラムを終了します.

• それ以外は150行のPLAY文で，チャネル1のFM音源で旋律部分を，チャネル2のFM音源で伴奏部分を演奏します.

```
100 ' ｼﾞｭｳｵﾝ ﾃﾞ ｴﾝｿｳ ｽﾙ
110 '
120 STOPM
130 PLAY "@13","@13"
140 READ M1$,M2$:IF M1$="end" THEN END
150 PLAY M1$,M2$:GOTO 140
160 '
170 ' Music data
180 '
190 DATA "T80L16V10"
195 DATA "T80L16V10"
200 '
210 DATA "O5ED#ED#E<B>DC<A8R16CEAB8R16EG#B>C8R16<E>ED#"
215 DATA "O2R8R4.A>EAR8.<E>EG#R8.<A>EAR8."
220 '
230 DATA "ED#E<B>DC<A8R16CEAB8R16E>C<BA4"
235 DATA "R4.<A>EAR8.<E>EG#R8.A>EAR16"
240 '
250 DATA "O5ED#ED#E<B>DC<A8R16CEAB8R16EG#B>C8R16<E>ED#"
255 DATA "O2R8R4.A>EAR8.<E>EG#R8.<A>EAR8."
260 '
270 DATA "ED#E<B>DC<A8R16CEAB8R16E>C<BA8R16B>CD"
275 DATA "R4.<A>EAR8.<E>EG#R8.A>EAR8."
280 '
290 DATA "O5E8.<G>FED8.<F>EDC8.<E>DC<B8R16E>ER8E>ER8<D#ER8D#"
295 DATA "O3CG>CR8.<<G>GBR8.<A>EAR8.<Q2EQ8>E>ER8E>ER8D#ER8D#ER16"
300 '
310 DATA "O5ED#ED#E<B>DC<A8R16CEAB8R16EG#B>C8R16<E>ED#"
315 DATA "O2R8R4.A>EAR8.<E>EG#R8.<A>EAR8."
320 '
330 DATA "ED#E<B>DC<A8R16CEAB8R16E>C<BA8R16B>CD"
335 DATA "R4.<A>EAR8.<E>EG#R8.A>EAR8."
340 '
350 DATA "O5E8.<G>FED8.<F>EDC8.<E>DC<B8R16E>ER8E>ER8<D#ER8D#"
355 DATA "O3CG>CR8.<<G>GBR8.<A>EAR8.<Q2EQ8>E>ER8E>ER8D#ER8D#ER16"
360 '
370 DATA "O5ED#ED#E<B>DC<A8R16CEAB8R16EG#B>C8R16<E>ED#"
375 DATA "O2R8R4.A>EAR8.<E>EG#R8.<A>EAR8."
380 '
390 DATA "ED#E<B>DC<A8R16CEAB8R16E>C<BA4"
395 DATA "R4.<A>EAR8.<E>EG#R8.<A4"
400 '
410 DATA "end",""
```

3.3 データファイルを作る

データファイルには，シーケンシャルファイルとランダムファイルの２種類があります．

シーケンシャルファイルとは，連続的に書かれたデータファイルのことで，書いた順序でしか読めないファイルです．その特徴は次のとおりです．

(1) データの書き込みおよび読み込みは，ファイルの先頭から順番に行います．

(2) ファイル内の一部のデータを書き換えることはできません．

(3) ファイルの変更は，ファイルの最後に新たにデータを追加することだけです．

(4) 比較的簡単に使えます．

(5) １つ１つのレコード(データの処理の単位)の大きさは，255文字以内で自由に変えられます．

それに対し，ランダムファイルはファイル内のデータ(レコード)を直接読み書きすることができます．その特徴は

(1) ファイル内の各レコードにランダムに読み書きできるので，データの書き込みおよび読み込みはファイルの先頭から順番に行われなくてもかまいません．

(2) ファイル内の一部のデータを書き換えることができます．

(3) ファイルの最後に新たにデータを追加することもできます．

(4) シーケンシャルファイルに比べてプログラムが複雑になります．

(5) １つ１つのレコードの大きさは256バイトに固定されています．

フロッピィディスクではシーケンシャルファイル，ランダムファイルの両方を取り扱うことができますが，カセットテープはシーケンシャルファイルのみを取り扱います．ここではカセットテープレコーダを使用する場合とフロッピィディスクドライブを使用する場合とにわけて説明します．

3.3.1 カセットテープレコーダを使用する場合

カセットテープレコーダを使用してデータの書き込み，読み込みを行う場合は，リモート端子(黒色)を接続します．

リモート端子を接続しますと，テープレコーダの ON/OFF は PC-8001MKⅡSR がコントロールします．ですから，PLAY ボタンを押していても，PC-8001MKⅡSRでカセットテープを使用する命令が実行されなければカセットテープは回りません．この機能によって，必要な間だけカセットテープを回すことができ，カセットテープのむだづかいが防げ，また手間もかかりません．

リモート端子を接続しなかった場合は，テープレコーダのスタート，ストップはすべて自

分でやる必要があります．ですから，カセットテープを使用する命令を実行する前にPLAYボタンを押し，終わるとSTOPボタンを押すという操作が必要になります．

データの書き込み，読み込みを行う場合は，リモート端子が接続されていないとうまく動作しない場合があります．以後リモート端子が接続されているものとして説明します．

1　データの書き込み

(1)　キーボードから次のプログラムを入力してください．

```
100 INPUT A,B,C,D
110 PRINT "Set tape,then push PLAY and REC"
120 INPUT "Ok";A$
130 PRINT #-1,A,B
140 PRINT #-1,C,D
150 END
```

このプログラムを簡単に説明しましょう．

- 100行ではキーボードから数値を入力します．
- 110行でメッセージが表示されますから，カセットテープをセットしRECボタン（またはSAVEボタン）とPLAYボタン（またはLOADボタン）を押して，120行で⏎キーを入力します．
- 130行と140行のPRINT#文で，カセットテープにデータを書き込みます．
- 150行でプログラムを終了します．

(2)　このプログラムを実行させます．

run ⏎

(3)　すると画面に"？"を表示します．これは100行のINPUT文がキーボードからの入力を要求しているのです．

そこで

1,2,3,4 ⏎

と入力します．

(4)　すると今度は次のメッセージが表示されます．

Set tape, then push PLAY and REC
Ok?

(5)　そこで，カセットテープをテープレコーダにセットします．このとき，テープカウンタを0にしておけば，あとでどこからデータが入っているか一目でわかり便利です．

(6)　カセットテープのセットが終わったら，テープレコーダのRECボタン（またはSAVEボタン）とPLAYボタン（またはLOADボタン）を押します．これらの準備が終わったら，

⏎

を入力します.

(7) すぐにカセットテープが回り始めます．しばらくしてから画面に"Ok"を出力し，カセットテープは止まります.

これでキーボードから入力した数値がカセットテープに書き込まれました.

2 データの読み込み

先程，カセットテープに書き込んだデータを今度は読み込んでみましょう.

(1) 次のプログラムを入力します.

```
100 PRINT "Push PLAY"
110 INPUT #-1,A,B
120 INPUT #-1,C,D
130 PRINT A,B,C,D
140 END
```

プログラムの意味は次のとおりです.

- 100行で，画面に"Push PLAY"と表示しPLAYボタン(またはLOADボタン)を押すことを促します.
- 110行と120行のINPUT#文で，カセットテープからデータを読み込みます.
- 130行で読み込んだデータを画面に表示します.
- 140行でプログラムを終了します.

(2) プログラムが入力できたら，データの書き込みを始めたところまでカセットテープを巻き戻します.

(3) カセットテープの準備ができたら実行させせます.

run⏎

(4) "Push PLAY"というメッセージが表示されたらPLAYボタン(またはLOADボタン)を押します．カセットテープが回り始めてしばらくすると，読み込まれたデータがディスプレイに表示されます.

3 データの型と数の一致

カセットテープにデータを書き込む，あるいは読み込む場合の注意事項として，書き込むときと読み込むときの〈データの型と数の一致〉があります．データとして文字を書き込んだのに，それを数値データとして読み込むことはできません．また，1つのPRINT#文で4つのデータを書き込んでいるのに，INPUT#文では2つしか読み込まないというわけにもいきません．ただし，データの型と数が一致していれば，読み込んだデータを代入する変数は，書き込

んだときに使った変数と異なっていてもかまいません．

〔良い例１〕

書き込むとき

```
100 PRINT #−1,A,B,C
```

読み込むとき

```
150 INPUT #−1,X,Y,Z
```

データの型(数値)，数(３つずつ)が一致しているのでOKです．

〔良い例２〕

書き込むとき

```
100 PRINT #−1,A$,B$
```

読み込むとき

```
150 INPUT #−1,C$,D$
```

これも，データの型(文字)，数(２つずつ)が一致しているのでOKです．

〔悪い例１〕

書き込むとき

```
100 PRINT #−1,A,B,C,D
```

読み込むとき

```
200 INPUT #−1,A,B
210 INPUT #−1,C,D
```

データの型(数値)は一致していますが，1つのPRINT#文で４つのデータを書き込んだのに対し，読み込むときは一度に２つしか扱っていません．この場合，200行で４つデータを読み込み最初の２つのデータを変数に代入しますが,残りの２つのデータは無視され"?Extra ignored"が表示されます．そして210行では読み込むデータを探し続けます．

〔悪い例２〕

書き込むとき

```
100 PRINT #−1,A$,B$
```

読み込むとき

```
200 INPUT #−1,A,B
```

データの数(２つ)は一致していますが,データの型が一致していません．この場合，200行で"Bad File Data"エラーが表示されます．

3.3.2 フロッピィディスクドライブを使用する場合

1 シーケンシャルファイル

シーケンシャルファイルを読み書きする際に使用できるステートメントと関数は次のとおりです.

文	OPEN CLOSE PRINT # PRINT # USING INPUT # LINE INPUT #
関数	INPUT$ EOF LOC LOF

シーケンシャルファイルを作り，その中のデータを読み書きするために必要な手順は次のようになります.

シーケンシャルファイルの作成

(1) 書き込みのためのファイルをオープンします.

例) open "TEST" for output as #1

(2) PRINT#文を使ってファイルにデータを書き込みます.

例) print #1, n$; ","; p

(3) 必要なだけ(2)の作業を繰り返した後，ファイルをクローズして終了します.

例) close #1

例としてキーボードから生徒のテストの点数を入力して，シーケンシャルファイル"TEST"を作るプログラム1と実行例を示します.

```
100 ' ﾌﾟﾛｸﾞﾗﾑ1
110 OPEN "TEST" FOR OUTPUT AS #1
120 '
130 INPUT "ﾅﾏｴ";N$
140 IF N$="end" THEN 200
150 INPUT "ｶﾓｸ";K$
160 INPUT "ﾄｸﾃﾝ";P
170 PRINT #1,N$;",";K$;",";P
180 PRINT
190 GOTO 120
200 '
```

```
210 CLOSE #1:END

run
ナマエ? テラダ゛ ユカリ
カモク? エイゴ゛
トクテン? 71

ナマエ? ナカハラ ミトシ
カモク? エイゴ゛
トクテン? 43

ナマエ? カネコ ユウコ
カモク? エイゴ゛
トクテン? 45

ナマエ? アイカワ フサコ
カモク? エイゴ゛
トクテン? 62

ナマエ? end
Ok
```

シーケンシャルファイルを作成するプログラムにおいて，PRINT#USING文を使用することによって，書式制御したデータを書き込むこともできます．たとえば，

例)　　print #1, using "####.##,";a,b,c,d

を使うと，特に区切り記号を付けることなく，フロッピィディスクに数値データを書き込むことができます．この区切り記号とは，シーケンシャルファイルの中のデータの読み込みにINPUT#文を使用するとき，数値データの場合はコンマ，キャリッジリターン，ラインフィード，空白などを区切りとしてデータを読み込むためのものです．LINE INPUT#文を使用する場合は，キャリッジリターンのみを区切り記号とします．

LOC関数をシーケンシャルファイルについて使用する場合は，そのファイルが開かれて以降，読み書きされた総セクタ数を値とします．

シーケンシャルファイルからの読み込み

(1)　読み込みのためのファイルをオープンします．

例)　　open "TEST" for input as #1

(2)　INPUT#文，LINE INPUT#文等を使ってファイルからデータを読み込みます．

例)　　input #1, n$, p

(3)　必要なだけ(2)の作業を繰り返した後，ファイルをクローズして終了します．

例)　　close #1

次にプログラムとして，プログラム1で作ったファイル"TEST"からデータを読み込み，50点未満の生徒を全部表示する例を示します．

```
100 ' ﾌﾟﾛｸﾞﾗﾑ2
110 OPEN "TEST" FOR INPUT AS #1
120 '
130 INPUT #1,N$,K$,P
140 IF P>=50 THEN 120
150 PRINT N$,K$,P
160 GOTO 120

run
ﾅｶﾊﾗ ﾐﾄｼ        ｴｲｺﾞ            43
ｶﾈｺ ﾕｳｺ         ｴｲｺﾞ            45
Input past end in 130
Ok
```

プログラム2ではファイル中のデータを最初から順番に読み込んでいきます．そして，ファイルの最後でもう読み込むデータがなくなると，130行で"Input past end"エラーが起こります．これを避けるために，125行に

```
125 IF EOF(1) THEN 200
```

を挿入してファイルの終わりをチェックし

```
200 CLOSE #1
210 END
```

のように処理します．

シーケンシャルファイルへのデータの追加

フロッピィディスク上にシーケンシャルファイルが既に作られていて，その後にデータを追加したい場合には，次のような手順で行います．

(1) 追加するためのファイルをオープンします．

例)　　open "TEST" for append as #1

(2) PRINT#文を使ってファイルに追加したいデータを書き込みます．

例)　　print #1, n$; "," ; p

(3) 必要なだけ(2)の作業を繰り返した後，ファイルをクローズして終了します．

例)　　close #1

プログラム1で作ったファイル"TEST"にデータを追加する例をプログラム3として示します．

```
100 ' ﾌﾟﾛｸﾞﾗﾑ3
110 OPEN "TEST" FOR APPEND  AS #1
120 '
130 INPUT "ﾅﾏｴ";N$
140 IF N$="end" THEN 200
150 INPUT "ｶﾓｸ";K$
```

```
160 INPUT "ﾄｸﾃﾝ";P
170 PRINT #1,N$;",";K$;",";P
180 PRINT
190 GOTO 120
200 '
210 CLOSE #1:END

run
ﾅﾏｴ? ﾅｶﾊﾗ ﾐﾄｼ
ｶﾓｸ? ﾘｶ
ﾄｸﾃﾝ? 99

ﾅﾏｴ? ｶﾈｺ ﾕｳｺ
ｶﾓｸ? ﾘｶ
ﾄｸﾃﾝ? 88

ﾅﾏｴ? ﾃﾗﾀﾞ ﾕｶﾘ
ｶﾓｸ? ｻﾝｽｳ
ﾄｸﾃﾝ? 44

ﾅﾏｴ? ｱｲｶﾜ ﾌｻｺ
ｶﾓｸ? ｼｬｶｲ
ﾄｸﾃﾝ? 41

ﾅﾏｴ? end
Ok
```

このプログラムは，プログラム１の110行のOUTPUTをAPPENDに変えただけです．したがって，このプログラムでデータを追加したファイルに対してプログラム２を実行してチェックすることができます．

```
run
ﾅｶﾊﾗ ﾐﾄｼ        ｴｲｺﾞ            43
ｶﾈｺ ﾕｳｺ         ｴｲｺﾞ            45
ﾃﾗﾀﾞ ﾕｶﾘ        ｻﾝｽｳ            44
ｱｲｶﾜ ﾌｻｺ        ｼｬｶｲ            41
Ok
```

シーケンシャルファイルのデータの修正

シーケンシャルファイルでは，そのファイルのデータを修正することができません．したがって，現在のファイルから読み込んで修正したデータを新しい別のシーケンシャルファイルへ書き込んでいく作業を行わなければなりません．以下にプログラム４として，プログラム１や３で作ったファイルの修正を行う例を示します．

```
100 ' ﾌﾟﾛｸﾞﾗﾑ4
110 OPEN "TEST" FOR INPUT AS #1
120 OPEN "test" FOR OUTPUT AS #2
130 '
140 IF EOF(1) THEN 500
150 INPUT #1,N$,K$,P
160 IF FLG=1 THEN 400
170 PRINT N$,K$,P
180 '
190 PRINT "ﾍﾝｺｳｶﾞ ｱﾘﾏｽｶ?";
200 INPUT "(yes or no or delete or end)";A$
```

```
210 A$=LEFT$(A$,1)
220 IF A$="e" THEN FLG=1:GOTO 400
230 IF A$="d" THEN 130
240 IF A$="n" THEN PRINT:GOTO 400
250 IF A$<>"y" THEN 180
260 INPUT "ﾅﾏｴ";N$
270 INPUT "ｶﾓｸ";K$
280 INPUT "ﾄｸﾃﾝ";P
290 PRINT
400 '
410 PRINT #2,N$;",";K$;",";P
420 GOTO 130
500 '
510 CLOSE #1
520 CLOSE #2
530 KILL "TEST"
540 NAME "test" AS "TEST"
550 END
```

上のプログラムでは，ファイル"TEST"から読み込んだデータに修正がない場合はそのまま新しいファイル"test"へ書き込みます．もし修正がある場合は，260～280行で新しいデータを入力し"test"へ書き込みます． 削除する場合には書き込まずに次のデータを"TEST"から読み込みます．修正が終了した場合は，"TEST"から読み込んだデータをそのまま"test"へ書き込む作業を，"TEST"のファイルの終わりまで繰り返します．

最後に，古いファイル"TEST"を消去し，"test"の名前を"TEST"に変更してから終了します．

2 ランダムファイル

ランダムファイルを読み書きする際に使用できるステートメントと関数は次のとおりです．

文	OPEN CLOSE FIELD LSET RSET PUT # GET #
関数	LOC LOF CVI / CVS / CVD MKI$ / MKS$ / MKD$

ランダムファイルを読み書きする場合には，次の手順が必要となります．

(1) ランダムファイルをオープンします．

例） open "TESTR" as #1

(2) ランダムファイルバッファに文字変数を割り当てます.

例)　　field #1,20 as n$,20 as k$,2 as p$

(3) データを書き込む場合は，その前にLSET文またはRSET文を使って，バッファへデータを転送しておきます.

例)　　lset n$=a$

またデータが数値の場合は，一度MKI$, MKS$, MKD$関数を使用して，それぞれ整数値，単精度実数値，倍精度実数値を2バイト，4バイト，8バイトの文字に変換してからバッファへ転送します.

例)　　lset p$=mki$(p)

(4) データの書き込みまたは読み込みを行います.

例)　　put #1,nm%

　　　　get #1,nm%

GET#文によって読み込んだデータが数値である場合には，CVI, CVS, CVD関数を使って文字列から数値へ変換します.

例)　　p=cvi(p$)

(5) ランダムファイルをクローズして作業を終了します.

例)　　close #1

次のプログラム5は，生徒番号によってランダムに読み書きするものです. 新しくファイルを作る場合はRUNコマンド実行後に"ファイルメイ"を入力したら，次に必ず"ファイル　ノ　イニシャライズ"を実行してください.

```
100 ' プログラム5
110  CLS:INPUT "ファイル メイ   ";F$
120  OPEN F$ AS #1
130  FIELD #1,1 AS ID$,20 AS N$,20 AS K$,2 AS P$
140 ' メニュー
150  CLS
160  PRINT "  1:データ サクセイ,ヘンコウ"
170  PRINT "  2:データ サンショウ"
180  PRINT "  3:テンスウ ニ ヨル ケンサク"
190  PRINT "  4:シュウリョウ"
200  PRINT "999:ファイル ノ イニシャライズ"
210  INPUT "  エランデ クダサイ";F
220   IF F=999 THEN 760
230  ON F GOTO 280,250,580,870
240 GOTO 140
250 ' データ サンショウ
260  SF=-1
270 GOTO 300
280 ' データ ヘンコウ
290  SF=0
300 '
310  GET #1,1
```

```
320  MAX=CVI(P$)
330  PRINT "ｾｲﾄ ﾊﾞﾝｺﾞｳ (1-";USING "###";MAX-1;
340  INPUT " or 0:ｵﾜﾘ)";NM%
350   IF NM%=0 THEN 140
360  NM%=NM%+1
370  GET #1,NM%
380   IF ID$=CHR$(255) AND SF=0 THEN 470
390   IF ID$=CHR$(255) AND SF THEN 300
400  P%=CVI(P$)
410  PRINT
420  PRINT "ﾅﾏｴ :";N$
430  PRINT "ｶﾓｸ :";K$
440  PRINT "ﾄｸﾃﾝ :";P%
450  PRINT
460   IF SF THEN 300
470 ' ｼﾂﾓﾝ
480  INPUT "ﾅﾏｴ ";A$
490   IF A$<>"" THEN LSET N$=A$
500  INPUT "ｶﾓｸ ";A$
510   IF A$<>"" THEN LSET K$=A$
520  INPUT "ﾄｸﾃﾝ ";A$
530   IF A$<>"" THEN LSET P$=MKI$(VAL(A$))
540  PRINT
550  LSET ID$="U"
560  PUT #1,NM%
570 GOTO 280
580 ' ｹﾝｻｸ
590  GET#1,1
600  MAX=CVI(P$)
610  INPUT "ﾅﾝﾃﾝｲｶ ﾃﾞ ｹﾝｻｸ ｼﾏｽｶ ";AK
620  PRINT
630  FOR NM%=2 TO MAX
640   GET #1,NM%
650    IF ID$=CHR$(255) THEN 690
660    IF CVI(P$)>AK THEN 690
670   PRINT USING "### ";NM%-1;
680   PRINT N$;K$;USING " ####";CVI(P$)
690 '
700  NEXT NM%
710  PRINT
720  PRINT "ｹﾝｻｸ ｼｭｳﾘｮｳ｡ RETURN ｦ ｵｼﾃ ｸﾀﾞｻｲ"
730 '
740  A$=INKEY$:IF A$="" THEN 730
750 GOTO 140
760 ' ｲﾆｼｬﾗｲｽﾞ
770  INPUT "ｾｲﾄﾊﾞﾝｺﾞｳ ﾉ ｻｲﾀﾞｲﾁ ﾊ";MAX
780  INPUT "ﾌｧｲﾙ ｦ ｲﾆｼｬﾗｲｽﾞ ｼﾏｽｶ (y or n) ";A$
790   IF A$<>"y" THEN 140
800  FOR NM%=1 TO MAX+1
810   LSET ID$=CHR$(255)
820   PUT #1,NM%
830  NEXT NM%
840  LSET P$=MKI$(MAX+1)
850  PUT #1,1
860 GOTO 140
870 ' ｼｭｳﾘｮｳ
880  CLOSE #1
890 END
```

3.4 デバッグ

プログラムが思ったとおりに動作しないことを「プログラムにバグ(虫)がいる」といい，それを直すことをデバッグ(虫とり)といいます．ここでは，BASICでプログラムを作ったときのデバッグの方法について説明します．

3.4.1 TRONコマンドを使ってのデバッグ

デバッグの際注目すべき点として，プログラムの実行順序と変数の内容の変化の2つが考えられます．

まずプログラムの実行順序を調べてみましょう．これを行うにはTRONコマンドを実行します．TRONコマンドはダイレクトモード，プログラムモードの両方で実行できます．TRONコマンドを実行すると，プログラムが実行された順に行番号が表示されますので，それによってプログラムの実行順序を調べることができます．また ESC キーを押すと，プログラムの実行を一時停止し，もう一度 ESC キーを押すと再開することができます．

このようにして，実行順序がおかしい行を見つけたら，その1つ手前の行の内容をLISTコマンドによって表示させます．そして，その行で参照されている変数の内容の変化が指定したとおりになっているかを調べます．これを行うには次のように，その変数に代入が行われた行(必ずしもLISTコマンドによって表示された行とは限りません)の前後にPRINT文を挿入します．

例)
```
       :
 90 B=B+0.1
100 A=SIN(B)          ←――― 注目している行
110 CSC(A)=1/SIN(A)
       :
```

上記のプログラムに次のようなPRINT文を挿入します．

```
 95 PRINT "debug line #95;";A,B
105 PRINT "debug line #105;";A,B
```

このように変更したプログラムを実行して，変数の内容が正しく変化しているか調べます．

数値変数の場合，一度も代入が行われなかったときの内容は0になります．したがって，上記の挿入文によって変数の内容を表示したとき，0になっているものがあれば，表示以前の行で代入が正しく行われたか確認してください．

上記のような現象は，変数名にミススペルがあった場合によく起こります．

間違いを見つけ，変更することによって削除したい行がある場合は，DELETEコマンドを使

わずREM文(')を使用します．この理由は，もしその変更が間違いとわかったとき，元に戻すのを簡単にするためです．

例)
```
100 A=SIN(B)
      ↓
100 'A=SIN(B)
```

また，挿入したい行がある場合は，その行を挿入し，その後に新規追加した行である旨をREM文(')を使ってコメントとして残しておくと便利です．

例)
```
100 A=SIN(B)
110 CSC(A)=1/SIN(A)
      ↓
100 A=SIN(B)
105 A=A+1 'bug fix
110 CSC(A)=1/SIN(A)
```

このようにしてデバッグを行い，完成したら，不要となった行やコメントを取り除きます．しかし，実行速度やプログラムサイズがそのプログラムに悪影響を与えない場合は，コメントは残しておいた方が後の変更やデバッグのために便利です．

3.4.2 スイッチを使ってのデバッグ

もっとデバッグしやすくするためには，プログラムそのものをデバッグしやすいように作ります．そのために，スイッチと呼ばれる変数を用意します．変数の値を0あるいは−1に設定し，その変数の値を切り替えることによってプログラムの実行順序を変えます．

スイッチはプログラムの最初の行で設定します．そしてプログラムの実行により一度代入されると，その値は以後変化しません．

たとえばDGというスイッチを用意して，DG=−1のとき変数を表示するようにプログラムを作ると，デバッグを行うのに便利です．そしてデバッグが終わったらDG=0にしておきます．

またスイッチは，デバッグが終わっても，後でバグが見つかったときのために残しておいた方がよいでしょう．

例)
```
10 FA=0
20 TR=NOT FA
30 DG=TR 'debug if true
      :
```

```
 95 IF DG THEN PRINT "debug line #95;";A,B
100 A=SIN(B)
105 IF DG THEN PRINT "debug line #105;";A,B
          ⋮
```

3.4.3 デバッグのために

デバッグを行う際の注意すべき点，参考点をあげました．

(1) プログラムの見やすさを重視してマルチステートメントは避ける．

```
10 INPUT "A=";A:INPUT "B=";B
20 C=A*B:PRINT C
          ↓
10 INPUT "A=";A
20 INPUT "B=";B
30 C=A*B
40 PRINT C
```

(2) 演算子の優先順位に気を付ける．（自信がないときはとりあえず( )を付ける.）

(3) 整数化する際のまるめ方に注意する．（小数点以下は切り捨てられます.）

(4) コロン(:)とセミコロン(;)，ピリオド(.)とコンマ(,)，ゼロ(0)とオー(O)，エル(l)とイチ(1)が間違っていないか注意を払う．

(5) 必要に応じてSTOP文を入れ，プログラムの実行順序や変数の内容を確認する．（CONTコマンドを使用すると，STOP文の次の行から実行が再開できます.）

(6) ON ERROR GOTO文を使ったプログラムをデバッグする場合，ON ERROR GOTO文が実行される状態だとエラー処理を行う必要が生じるため，デバッグがやりにくくなる．デバッグ中はON ERROR GOTO文をコメント文(REM文を使用)に変更しておく．

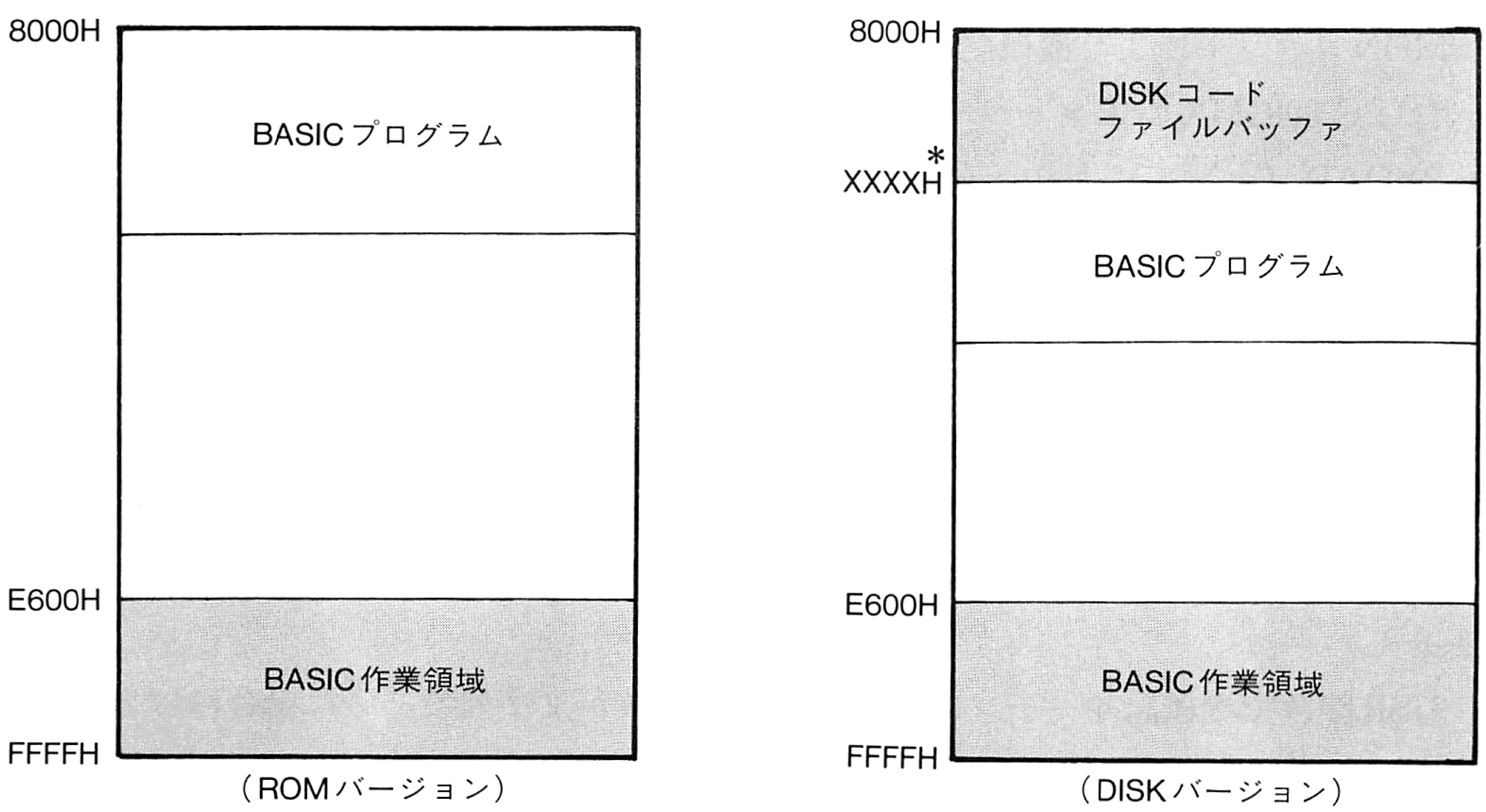
3.5　機械語プログラムの呼び方

N_{80SR}-BASICには，機械語で書かれたプログラムを呼び出す機能としてUSR関数が用意されています．ここでは機械語プログラムを作成する際の注意と，USR関数との引数の受け渡し方について説明します．なお，ここでの説明を正しく理解するためには，μPD780のプログラミングに関する知識を必要とします．

3.5.1　メモリの配置

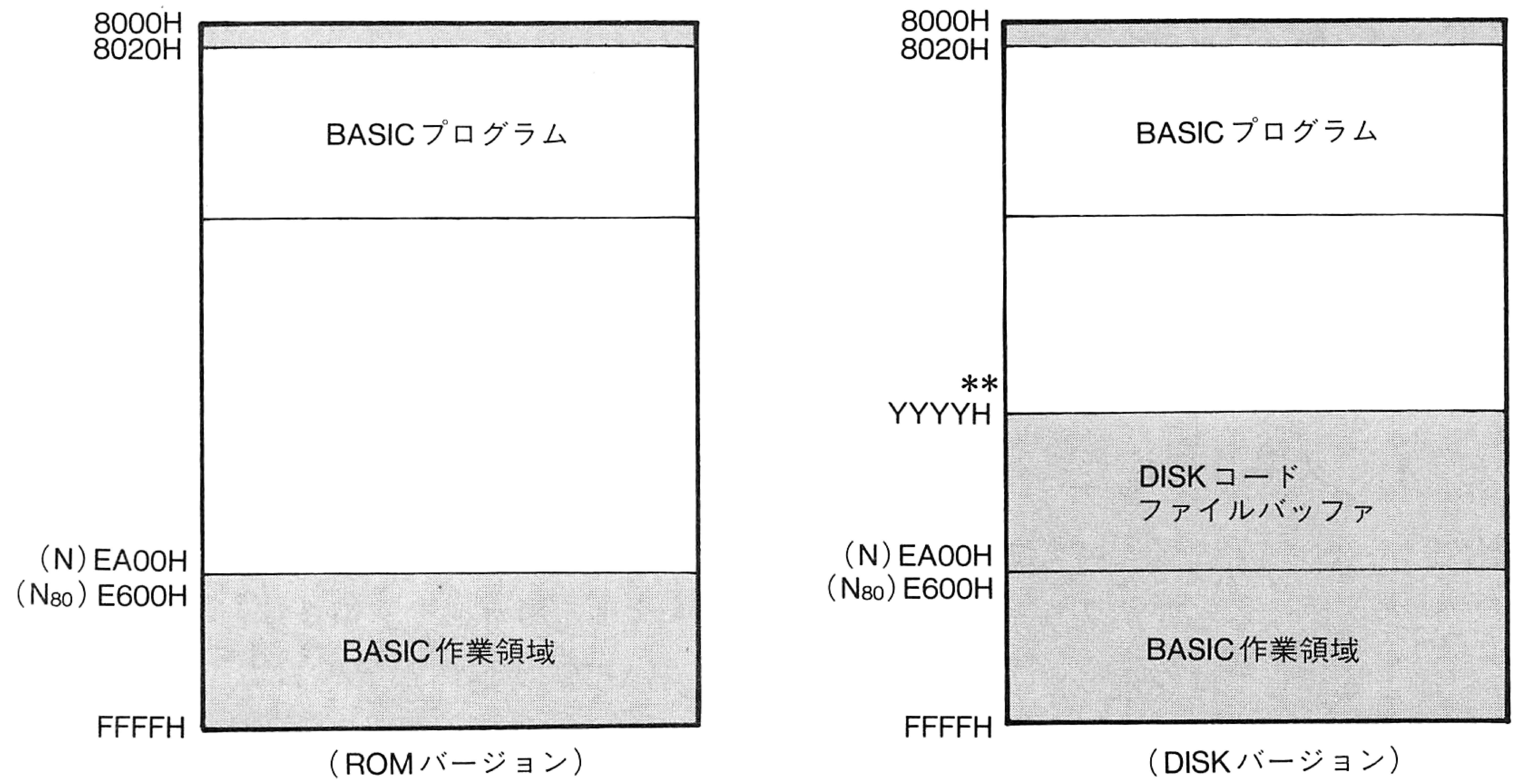
機械語ルーチンを機械語モニタにより作成したり，ファイルからロードする際には，そのためのメモリ領域を確保する必要があります．この機械語プログラム用のメモリ領域は，BASICの使用する領域と重複してはなりません．そうでなければ，BASICの動作が異常になったり，逆にBASICの動作により準備した機械語プログラムが破壊されたりします．メモリの確保はCLEAR文によりBASICの使用するメモリの上限を設定することにより行います．指定された上限からBASICの作業領域の前まではBASICは利用しなくなりますので，機械語プログラム領域として用いることができます．BASICの作業領域とは，BASIC自身が使用している領域のことでユーザが使用することはできません．

参考：ユーザがBASICプログラムや機械語プログラムで使用可能なメモリの領域は次のメモリマップ上でアミ（　）のかかっていない部分です．それ以外の領域を変更した場合BASICの動作は保障されません．



＊　XXXXH＝peek(&heae6)＊256＋peek(&heae5)

N_{80SR}-BASICのメモリマップ



** YYYYH=peek(&hef55)*256+peek(&hef54) (システム起動時の値)

N_{80}-BASIC, N-BASICのメモリマップ

3.5.2 機械語プログラムの準備

確保した領域に機械語プログラムを準備する主な方法としては，次の3つがあります．

(1) 機械語モニタを用いる

MONコマンドにより機械語モニタに移り，そのモニタの機能を利用してプログラムを書き込む(PC-8001MKⅡSRユーザーズガイド参照)．

(2) BLOADコマンドによりロードする

既にBSAVEコマンドによりセーブされている機械語プログラムをロードする．

(3) POKE文でプログラムを書き込む

比較的短いプログラムであれば，DATA文により機械語プログラムをデータの型で表現しておき，それをREAD文で読み出した後，POKE文により書き込む．

また，機械語プログラムを作成する際には次の点に注意してください．

(a) BASICに制御を戻すためには，プログラムはリターン命令(RET)を用いなければなりません．

(b) USR関数で機械語ルーチンに引き渡された引数が文字列であった場合には，〔DE〕レジスタペアに次のような3バイトからなるストリング ディスクリプタと呼ばれる領域の，先頭番地が入ります．

- ストリングディスクリプタの最初のバイトは，その文字列の長さ(0から255)を保持

します．

・2番目のバイトは，文字列の置かれている先頭番地の下位8ビットを保持します．

・最後のバイトは，文字列の置かれている先頭番地の上位8ビットを保持します．

機械語ルーチンでは，この3バイトのストリングディスクリプタを書き換えてはなりません．また，ストリングディスクリプタによって指されるストリングの内容は変更することはできますが，その長さを変更してはなりません．

(c) 機械語ルーチンからBASICに戻るときには，呼び出されたときとスタックポインタ(SP)の値が等しくなっていなければなりません．

(d) 機械語ルーチンが呼び出されたとき，スタック領域としては8レベル(16バイト)分だけ利用できるように設定されています．もし機械語ルーチン内でより大きなスタックを必要とする場合には，ルーチン内で独自にスタックを用意しなければなりません．またこの場合，BASICに戻るときにスタックポインタを戻すことを忘れてはなりません．

3.5.3 USR関数の呼び出し

USR関数は次のような書式で使います．

USR〔〈番号〉〕(〈引数〉)

ただし〈番号〉は0から9までの数値で，引数は任意の数値式または文字式です．USR関数は書式上〈引数〉を必要としますので，呼び出されたルーチンが引数を必要としない場合でも，ダミーの引数を与える必要があります．

USR関数が呼び出され，対応する機械語ルーチンに制御が移ったとき，Aレジスタは，渡された引数の型を示す値を持っています．その値と意味は次のようになっています．

Aの値	引数の型
2	整数型
3	文字列型
4	単精度型
8	倍精度型

引数が文字列の場合には，〔DE〕レジスタペアは3バイトから成るストリングディスクリプタを指しています．

引数が数値の場合には，〔HL〕レジスタに浮動小数点アキュームレータ(FAC)と呼ばれる8バイトの領域の5バイト目のアドレスが入っています．実際の引数は，このFACの内に各型に応じて次のように格納されて渡されます．

・整数型のとき

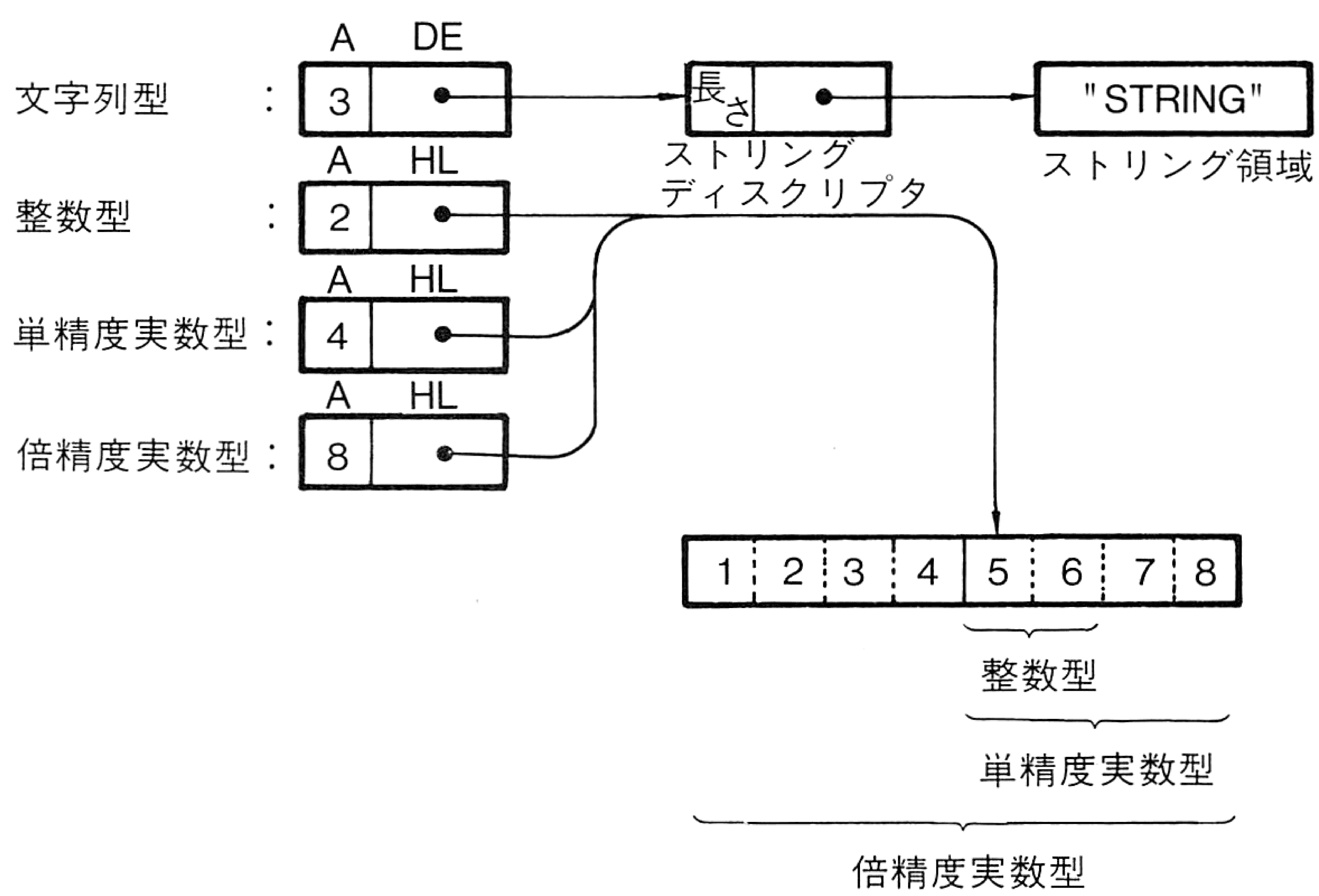
FACの5バイト目(〔HL〕レジスタペアが指しているところ)に引数の下位8ビット，6バイト目に上位8ビットが入ります．

・単精度実数型のとき

FACの8バイト目は指数部になっており，(指数−128)の値が入ります．小数点は仮数部の最上位ビットの左にあると想定します．7バイト目は仮数部の最高部7ビットを保持します．このバイトの最上位ビットは符号を示し，0で正，1で負を表します．6バイト目と5バイト目はそれぞれ仮数部の中部と最低部の8ビットを保持します．

・倍精度実数型のとき

5バイト目から8バイト目までは単精度実数型と同じように指数部と仮数部の上位3バイトが入ります．1バイト目から4バイト目には仮数部の下位4バイトが入ります．



浮動小数点アキュームレータ(FAC)

USR関数がBASICに値を返すには，関数値をFACに設定してやることで行えます．通常，返す値は，引数として渡した値と同じ型となります．

次にUSR関数を使ったサンプルプログラムとして，UFOを打ち落とすゲームを作ってみました．スペースキーを押すとミサイルが発射されます．(このプログラムは，N_{80SR}-BASICで実行してください)

```
100 CLEAR 300,&HBFFF
110 DEFINT A-Y:DEFSTR Z:DIM Z$(1)
120 DEF USR=&HC000:GOSUB 420
130 Z(0)=" ⌒∨╮":Z(1)=" ╰∨╯"
140 ZS=" ╭◆╮":Z="|":ZC="    "
150 CONSOLE 0,25,0,1:WIDTH 80,25
```

```
160 COLOR 6,,32
170 PRINT CHR$(12)
180 N=0:K=38:L=39:A=23:B=24:S=1:W=0
190 LOCATE K,B:PRINT ZS;
200 LOCATE L,A:PRINT Z;
210 H=0
220 LOCATE H,N:PRINT Z(H MOD 2);
230 IF W THEN GOSUB 270:GOTO 250
240 IF INP(9)=&HBF THEN W=1 ELSE GOSUB 410
250 LOCATE H,N:PRINT ZC;
260 H=H+1:IF H>76 THEN 210 ELSE 220
270 A=A-1:IF A>=N THEN 310
280 A=23:W=0:LOCATE L,N:PRINT " ";
290 LOCATE L,A:PRINT Z;
300 RETURN
310 AD=&HF3F0+A*120:D=USR(AD):IF D THEN 330
320 RETURN
330 FOR I=0 TO 5
340 LOCATE L,N:PRINT "X";:BEEP 1:GOSUB 410
350 LOCATE L,N:PRINT " ";:BEEP 0:GOSUB 410
360 LOCATE L-1,N:PRINT "    ";
370 LOCATE L,N+1:PRINT " ";
380 NEXT I:A=23:H=77:W=0
390 LOCATE L,A:PRINT Z;
400 RETURN
410 FOR J=0 TO 10:NEXT J:RETURN
420 FOR AD=&HC000 TO &HC027
430 READ DA:POKE AD,DA
440 NEXT AD
450 RETURN
460 DATA &HFE,2          :'       CP      2
470 DATA &HC0            :'       RET     Z
480 DATA &H5E            :'       LD      E,(HL)
490 DATA &H23            :'       INC     HL
500 DATA &H56            :'       LD      D (HL)
510 DATA &HE5            :'       PUSH    HL
520 DATA &HEB            :'       EX      DE,HL
530 DATA &H7E            :'       LD      A,(HL)
540 DATA &HFE,&H76       :'       CP      'v'
550 DATA &HCA,&H1F,&HC0  :'       JP      Z,HIT
560 DATA &H36,&H96       :'       LD      (HL),'|'
570 DATA &H11,&H78,0     :'       LD      DE,120
580 DATA &H19            :'       ADD     HL,DE
590 DATA &H36,&H20       :'       LD      (HL),' '
600 DATA &HE1            :'       POP     HL
610 DATA &H36,0          :'       LD      (HL),0
620 DATA &H2B            :'       DEC     HL
630 DATA &H36,0          :'       LD      (HL),0
640 DATA &H3E,2          :'       LD      A,2
650 DATA &HC9            :'       RET
660 DATA &HE1            :'   HIT:POP     HL
670 DATA &H36,&HFF       :'       LD      (HL),&HFF
680 DATA &H2B            :'       DEC     HL
690 DATA &H36,&HFF       :'       LD      (HL),&HFF
700 DATA &H3E,2          :'       LD      A,2
710 DATA &HC9            :'
```

ここでは，機械語のプログラムは，DATA文で用意しておきPOKE文でC000H番地以降に書き込んでいます．

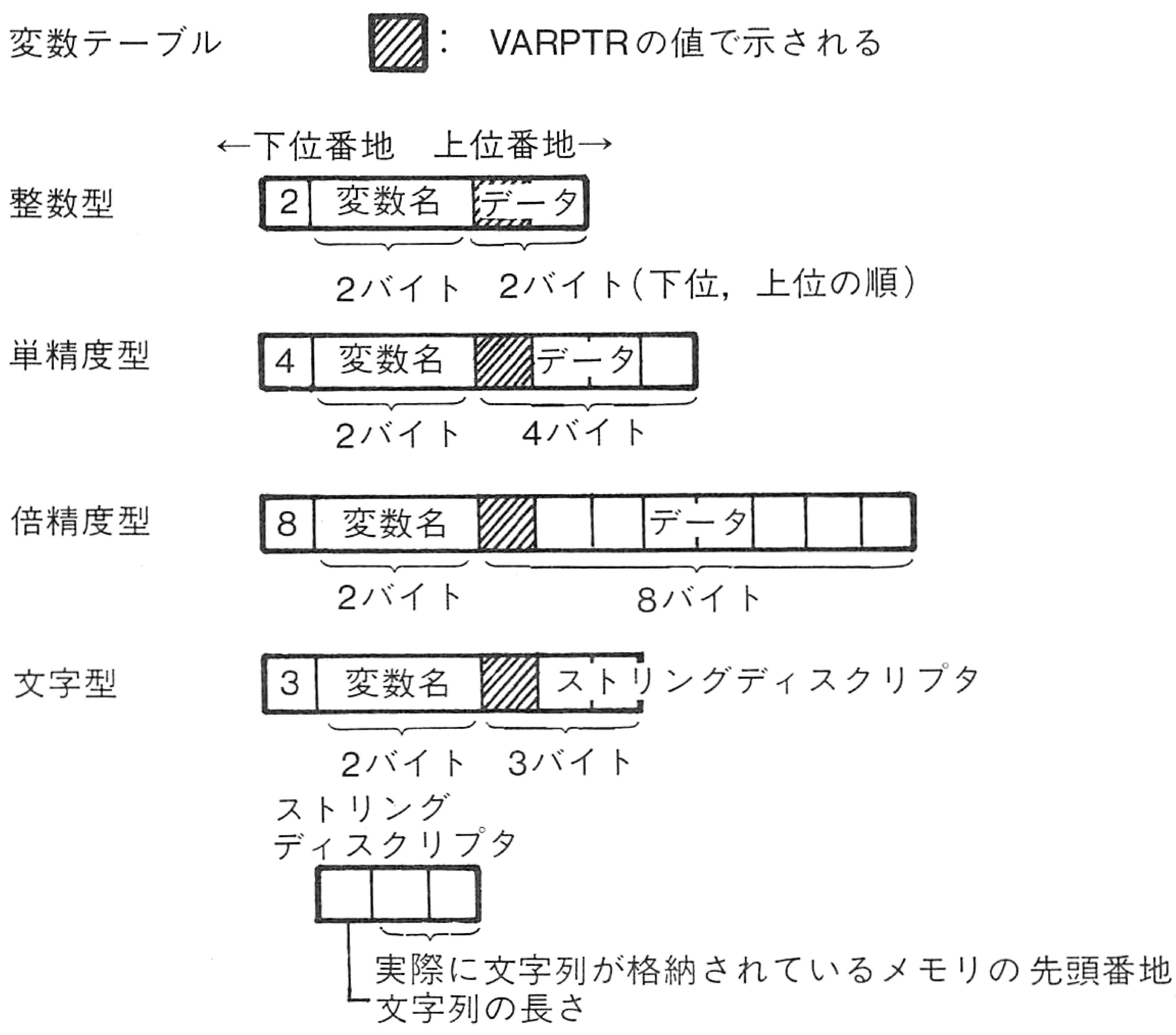
機械語ルーチンは，ミサイルの表示と，ミサイルがUFOに当たったのかどうかの判定を行います．310行でUSR関数の引数として，次にミサイルを表示する位置のVRAMのアドレスを渡しています．機械語ルーチンでは，そのアドレスにUFOがいるかどうか調べ，もしいれば－1，

いなければミサイルを移動して0を返します．

3.5.4 VARPTR

VARPTR関数は，指定された変数，バッファの置かれている番地を返します．

VARPTR関数の引数として数値変数名が与えられたときには，その変数の値の最下位番地を返します．上位バイトは下図に示すようにこの番地以降に入っています．また，文字変数名が指定された場合にはストリングディスクリプタの先頭番地が返されます．



変数として配列要素が指定された場合には，その要素の置かれている番地が返されます．

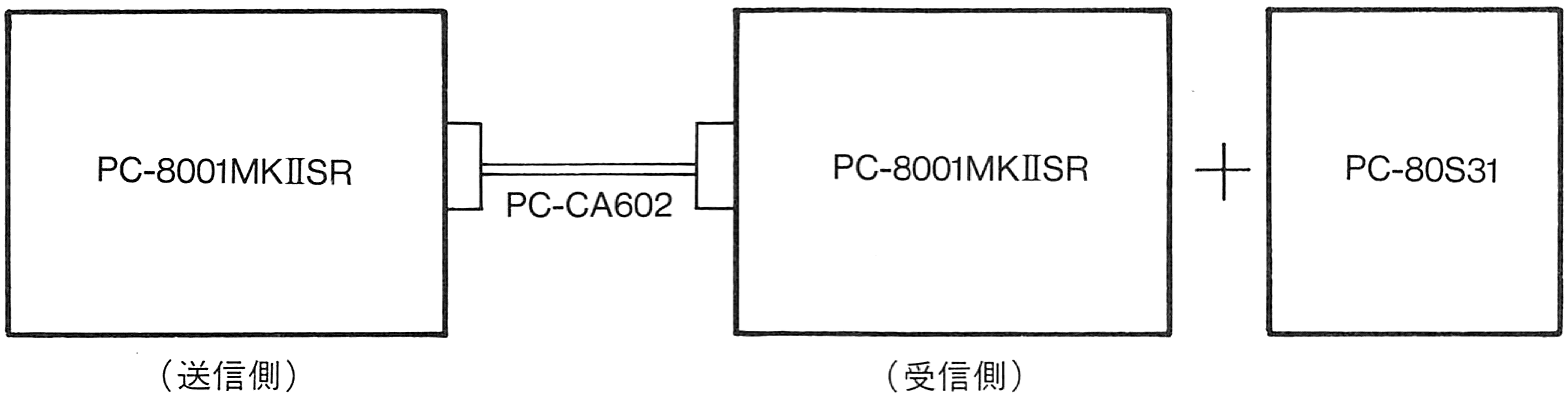
引数としてバッファ番号が与えられた場合には，そのバッファ領域の番地を返します．この領域の最初の9バイトはBASIC作業用の領域となっており，ファイルとの入出力に用いられる256バイトの領域は，VARPTR関数の値に9を加えた番地から始まります．

3.6 RS-232C通信ポートを使ったプログラム

PC-8001MKⅡSRは，通信機能としてRS-232C通信ポートを持っています．このポートは，音響カプラやタブレットなどの周辺装置や，別のRS-232C通信ポートを持つパーソナルコンピュータとの接続に使います．

ここでは，サンプルプログラムとして，2台のPC-8001MKⅡSRを使った電子メールプログラムを紹介します．

使用するシステムは次のとおりです．



2台のPC-8001MKⅡSRを準備します(1台は，フロッピィディスクユニット付き)．接続には，PC-CA602(RS-232Cケーブル(リバース))を使います．

ボーレートは，本体背面のジャンパスイッチで1200ボーに設定してください．

送信側のプログラム

```
100 ' ﾃﾞﾝｼ ﾒｲﾙ ﾌﾟﾛｸﾞﾗﾑ ( ｿｳｼﾝ )
110 INIT %1,&HFA,&H37
120 CLS:PRINT "ﾒｯｾｰｼﾞｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ｡"
130 LINE INPUT T$
140 PRINT %1,T$:BEEP
150 IF T$<>"end" THEN 130
160 END
```

これは，送信側で使用するプログラムです．

• 110行で，通信方式を決め，RS-232C通信ポートを初期化しています．
  通信方式は，
    • ストップビット＝2
    • パリティ＝偶数
    • キャラクタ長＝7ビット
  です．

• 130行から150行が，入力および出力のループです．130行で，キーボードから入力したメッセージを変数T$に代入しています．それをRS-232C通信ポートに送っているのが

140行です．送信し終わると短いブザーを鳴らします．メッセージとして"end"を入力すると，150行でこのループから抜けます．

受信側のプログラム

```
100 ' ﾃﾞﾝｼ ﾒｲﾙ ﾌﾟﾛｸﾞﾗﾑ ( ｼﾞｭｼﾝ )
110 INIT %1,&HFA,&H37
120 OPEN "mail.dat" FOR OUTPUT AS #1
130 LINE INPUT %1,T$
140 PRINT T$
150 PRINT #1,T$
160 IF T$<>"end" THEN 130
170 CLOSE
180 END
```

これは，受信側で使用するプログラムです．

• 110行のINIT％文は，送信側と同じモードを指定します．

• 120行は，受けたメッセージを格納するディスクファイルを開きます．

• 130行から160行が入力および出力のループです．130行でRS-232C通信ポートの入力バッファからメッセージを取り出し，変数T$に代入します．140行および150行では，そのメッセージを，画面とディスクファイルに書き出しています．そして160行で"end"を受けたと判断したらファイルを閉じて，プログラムを終了します．

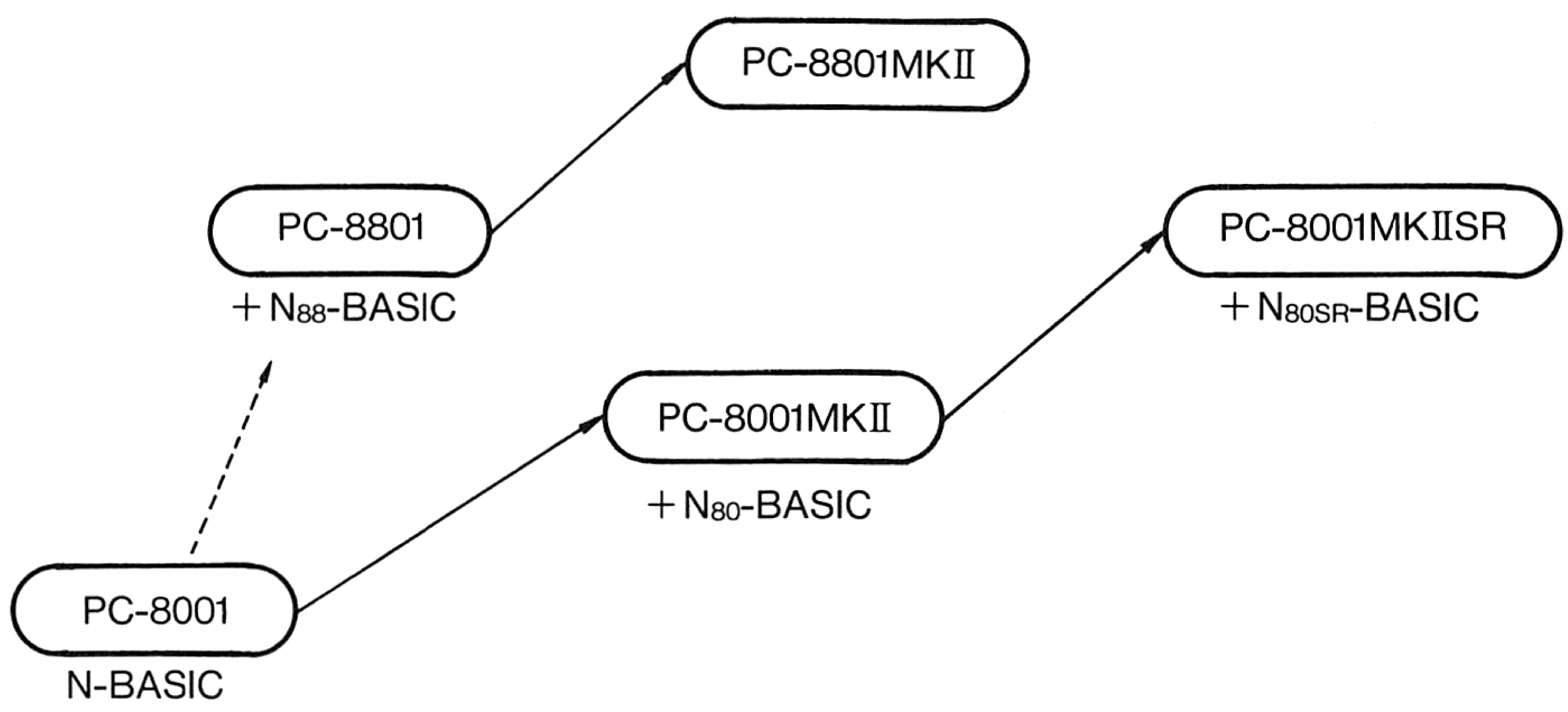
参照：INIT％文のパラメータの設定のしかたについては，資料6　μPD8251の使い方を参照してください．

資　料

資料１　PC-8000/8800シリーズでの位置付け

PC-8001MKⅡSRは，PC-8001MKⅡの後継機種として，そのハードウェア/ソフトウェアの機能を継承した上で，大幅な機能強化がなされたものです．

PC-8000/8800シリーズの中で見た位置付けは，次の図のようになります．



資料２　N_{80}-BASICとN-BASIC

資料２．１　N_{80SR}-BASICとの関係

PC-8001MKⅡSRは，N_{80SR}-BASICの他に，従来機種との互換性のため，N_{80}-BASIC（PC-8001MKⅡ で使用可）とN-BASIC（PC-8001, PC-8801, PC-8001MKⅡ, PC-8801MKⅡ で使用可）を搭載しています．

３つのBASICは，核の部分は共通ですから，一部の命令を除いて互換性があります．この関係を図で表すと次のようになります．

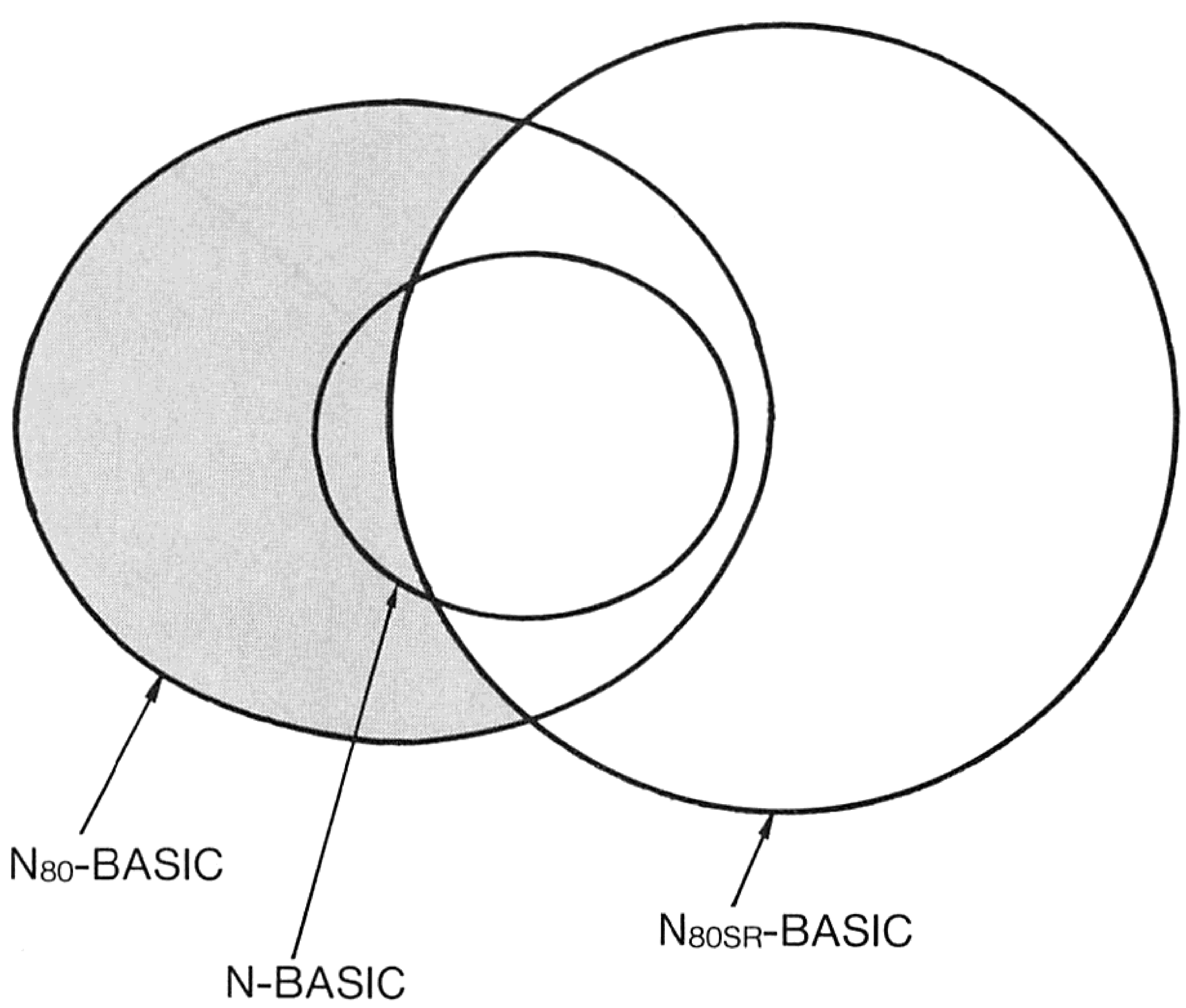


図2.1　３つのBASICの関係

１章～3 章ではN_{80SR}-BASICについて解説してきました．この節では，N_{80}-BASIC, N-BASICを使う人のために，N_{80SR}-BASICに含まれない命令（図のアミ　　部分）およびN_{80SR}-BASICとは機能が一部異なる命令について説明していきます．

資料2.2　N₈₀-BASIC/N-BASICコマンド·ステートメント·関数·予約変数一覧

N₈₀-BASICおよびN-BASICのコマンド・ステートメント・関数・予約変数を次に示します．

N₈₀-BASIC/N-BASICコマンド・ステートメント・関数・予約変数一覧表

コマンド・ステートメント・関数・予約変数	N_{80}	N
ABS	1	1
ASC	1	1
ATN	1	1
ATTR$	1	1
AUTO	1	1
BEEP	1	1
CDBL	1	1
CHR$	1	1
CINT	1	1
CLEAR	2	2
CLOAD	1	1
CLOAD?	1	1
CLOSE	1	1
CMD BLOAD	3	—
CMD BSAVE	3	—
CMD CIRCLE	3	—
CMD CLS	3	—
CMD COLOR	3	—
CMD COLOR@	3	—
CMD COPY	3	—
CMD GET@	3	—
CMD INIT	3	—
CMD LINE	3	—
CMD PAINT	3	—
CMD POINT	3	—
CMD PRESET	3	—
CMD PSET	3	—
CMD PUT@	3	—
CMD SCREEN	3	—
CMD VIEW	3	—
COLOR	2	2
CONSOLE	2	2
CONT	1	1
COS	1	1
CSAVE	1	1
CSNG	1	1
CSRLIN	1	1

コマンド・ステートメント・関数・予約変数	N_{80}	N
CVI/CVS/CVD	1	1
DATA	1	1
DATE$	1	1
DEF FN	1	1
DEFINT/SNG/DBL/STR	1	1
DEF USR	1	1
DELETE	1	1
DIM	1	1
DSKF	1	2
DSKI$	1	1
DSKO$	1	1
END	1	1
EOF	1	1
ERASE	1	1
ERL/ERR	1	1
ERROR	1	1
EXP	1	1
FIELD	1	1
FILES/LFILES	1	1
FIX	1	1
FOR…TO…STEP～NEXT	1	1
FORMAT	—	3
FPOS	1	1
FRE	1	1
GET #	1	1
GET@	3	3
GOSUB～RETURN	1	1
GOTO/GO TO	1	1
HEX$	1	1
IF…THEN…ELSE/IF…GOTO…ELSE	1	1
INIT%	2	2
INKEY$	1	1
INP	1	1
INPUT	1	1
INPUT #	1	1

コマンド・ステートメント・関数・予約変数	N_{80}	N
INPUT$	1	1
INPUT%	2	2
INSTR	1	1
INT	1	1
KEY	1	1
KEY LIST	1	1
KILL	1	1
LEFT$	1	1
LEN	1	1
LET	1	1
LFILES	1	1
LINE	3	3
LINE INPUT	1	1
LINE INPUT #	1	1
LINE INPUT%	2	2
LIST/LLIST	1	1
LOAD	1	1
LOC	1	1
LOCATE	1	1
LOF	1	1
LOG	1	1
LPOS	1	1
LSET/RSET	1	1
MERGE	1	1
MID$(ステートメント)	1	1
MID$(関数)	1	1
MKI$/MKS$/MKD$	1	1
MON	2	2
MOTOR	1	1
MOUNT	—	3
NAME	1	1
NEW	1	1
OCT$	1	1
ON ERROR GOTO	1	1
ON…GOSUB/ON…GOTO	1	1
OPEN	1	1

コマンド・ステートメント・関数・予約変数	N_{80}	N	コマンド・ステートメント・関数・予約変数	N_{80}	N	コマンド・ステートメント・関数・予約変数	N_{80}	N
OUT	1	1	READ	1	1	STATUS POINT	3	—
PEEK	1	1	REM	1	1	STATUS VIEW	3	—
POINT	3	3	REMOVE	—	3	STR$	1	1
POKE	1	1	RENUM	1	1	STRING$	1	1
PORT	2	2	RESTORE	1	1	STOP	1	1
POS	1	1	RESUME	1	1	SWAP	1	1
PRESET	3	3	RIGHT$	1	1	TAB	1	1
PRINT/LPRINT	1	1	RND	1	1	TAN	1	1
PRINT #	1	1	RUN	1	1	TERM	3	3
PRINT%	2	2	SAVE	1	1	TIME$	1	1
PRINT USING/ LPRINT USING	1	1	SET	1	1	TRON/TROFF	1	1
			SGN	1	1	USR	1	1
PRINT # USING	1	1	SIN	1	1	VAL	1	1
PSET	3	3	SPASE$	1	1	VARPTR	1	1
PUT #	1	1	SPC	1	1	WAIT	1	1
PUT@	3	3	SQR	1	1	WIDTH	2	2

1：N_{80SR}-BASICと同じ機能です．（使い方については第2章を参照してください．）

2：N_{80SR}-BASICと機能が一部異なります．（資料2.3　N_{80SR}-BASICと一部異なる機能・命令を参照してください．）

3：N_{80}-BASIC/N-BASIC専用命令です．（資料2.4　N_{80}-BASIC/N-BASIC専用命令を参照してください．）

“—”は，N_{80}-BASICあるいはN-BASICにその命令がないことを意味します．

資料２．３　N_{80SR}-BASICと一部異なる機能・命令

資料２．３．１　N_{80SR}-BASICと一部異なる機能

N-BASIC, N_{80}-BASICは低解像度グラフィック機能を持っています．このグラフィック機能はテキスト画面で操作することができ，グラフィック画面は使用しません．

低解像度グラフィックの解像度は，WIDTH文で設定された桁数と行数に依存します．

横　〈桁数〉×2ドット

縦　〈行数〉×4ドット

が低解像度グラフィックの解像度になります．

座標は画面の左上が原点(0，0)になり，最大値はX座標，Y座標とも必ず各解像度－1になります．

ただし，この低解像度グラフィックでは，相対座標による指定およびビューポートを設定することはできません．

資料２．３．２　N_{80SR}-BASICと一部機能が異なる命令

下記にあげた命令はN_{80SR}-BASICと一部機能が異なっているため，N_{80}-BASICおよびN-BASICでの機能や使い方を示したものです．N80マーク，Nマークはその命令の機能が違っているBASICを表したものです．

命　　令	BASIC	機　　　　能
CLEAR	N80 N	・〈ストリング領域の大きさ〉の初期値は300に設定されています． それ以外はN_{80SR}-BASICと同じ機能を持ちます．
COLOR	N80 N	・次の書式を用います． **COLOR〔〈ファンクションコード〉〕〔,〈ヌルキャラクタコード〉〕〔,〈グラフィックスイッチ〉〕** ・〈ファンクションコード〉，〈ヌルキャラクタコード〉の指定はN_{80SR}-BASICと同様です． ・〈グラフィックスイッチ〉は，低解像度グラフィックモードまたはキャラクタモードの選択を行います．1で低解像度グラフィックモード，0でキャラクタモードを設定します．
CONSOLE	N80 N	・書式は次のようになります． **CONSOLE〔〈スクロール開始行〉〕〔,〈スクロール行数〉〕〔,〈ファンクションキー表示スイッチ〉〕〔,〈カラー／白黒スイッチ〉〕** ・〈文字フォント〉機能がつかないことを除けば，その機能はN_{80SR}-BASICと全く同じです．

命　令	BASIC	機　　能
DSKF	N	・書式は次のようになります． **DSKF (〈ドライブ番号〉)** ・〈ドライブ番号〉で指定されたフロッピィディスクの残り容量をクラスタ単位で返します． ・N-BASIC DISKバージョンではミニフロッピィディスクだけしか使用できませんから1クラスタ＝8セクタ(2Kバイト)に固定されています．
RS-232C通信ポートを使う命令 INIT% INPUT% LINE INPUT% PORT PRINT%	N_{80} N	・N_{80}-BASICおよびN-BASICモードでRS-232C通信ポートを使う場合，次の注意が必要です． ① プログラムの先頭で次の文を実行してください． **poke &h8000, &h7f** **poke &h8001, &h23** **a=&h20 or peek(&hea66)** **poke &hea66, a** **out &h30, a** **out &he6, 4** **out &he4, &hff** ② プログラム終了時，次の文を実行してください． **out &he6, 0** **a=&hdf and peek(&hea66)** **poke &hea66, a** **out &h30, a**
MON	N_{80} N	・コマンドの種類が異なります． ・詳しくはユーザーズガイドを参照してください．
WIDTH	N_{80} N	・書式はN_{80SR}-BASICと同様です． ・指定できる〈桁数〉は80，72，40，36のいずれかであり，〈行数〉は25または20のいずれかになります．白黒モードあるいはカラーモードにした場合の機能はN_{80SR}-BASICのときと同じです． ・〈桁数〉あるいは〈行数〉のどちらかを省略した場合は，N_{80SR}-BASICと同様現在の〈桁数〉，〈行数〉が採用されます．〈桁数〉，〈行数〉の両方を省略して使うことはできません．

資料2.4　N_{80}-BASIC/N-BASIC専用命令

N_{80}-BASICおよびN-BASIC専用命令を

(1)　N_{80}-BASICとN-BASICの両方で使用することができる命令

(2)　N_{80}-BASICだけで使用する命令

(3)　N-BASICだけで使用する命令

の3つに分けて，その個々の命令について解説します．

グラフィックステートメント

GET@

ゲット・アットマーク：get @

機　能　テキスト画面上の任意の範囲のキャラクタ，または低解像度グラフィックパターンを配列へ読み込む

書　式

(1)　GET@(cx1 , cy1)－(cx2 , cy2),〈配列名〉

(2)　GET@(lx1 , ly1)－(lx2 , ly2),〈配列名〉, G

(3)　GET@A(cx1 , cy1)－(cx2 , cy2),〈配列名〉

解　説

•(1)の書式の場合，キャラクタ座標において(cx1, cy1)と(cx2, cy2)を対角線とする四角形の内側に表示されているキャラクタを配列へ読み込みます．この場合の水平位置cx1, cx2は0～(1行の桁数)－1までの値，垂直位置cy1, cy2は0～(1画面の行数)－1までの値で指定し，配列の添字の値は次のようになります．

〈添字の値〉＝(キャラクタ単位で計った面積/N)－1

Nは読み込む配列の型により異なり，整数型配列で2，単精度実数型配列で4，倍精度実数型配列で8となります．

•(2)の書式の場合，低解像度グラフィック座標において(lx1, ly1)と(lx2, ly2)を対角線とする四角形の内側に表示されている低解像度グラフィックパターンを配列に読み込みます．この場合の水平位置lx1, lx2は0～(1行の桁数)×2－1までの値，垂直位置ly1, ly2は0～(1画面の行数)×4－1までの値をとります．配列の添字の値は次のようになります．

〈添字の値〉＝(ドット単位で計った面積/8＋2)/N

Nはキャラクタの場合と同じです．

•(3)の書式は，キャラクタと低解像度グラフィックパターン，あるいは色や機能の異なる画面上のデータを配列に読み込みます．座標の指定は書式(1)の場合と同じで，配列の添字の値は次のようになります．

〈添字の値〉＝(キャラクタ単位で計った面積/N)＊2－1

Nはキャラクタの場合と同じです．

例　GET@(5,5)−(50,60),A%,G

• 低解像度グラフィック座標(5，5)～(50，60)の範囲にあるグラフィックパターンを配列A%に読み込みます．

プログラム例

```
list
100 ' GET@
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 DIM G%(100)
140 LINE(0,50)-(9,59),PSET,B
150 GET@(0,50)-(9,59),G%,G
160 FOR X=20 TO 140 STEP 20
170   PUT@(X,50)-(X+9,59),G%,PSET
180 NEXT X
190 END
Ok
```

• テキスト画面に四角形を8個描きます．

• 140行で，画面左端に1個四角形を描きます．

• 150行で，描いたパターンを配列G%に読み込みます．

• 160行から180行で，読み込んだパターンを7個出力します．

参照：PUT@

LINE

ライン：line

機　能　低解像度グラフィックで直線や長方形を描く
キャラクタで直線や長方形を描く

書　式

(1)　LINE(lx1, ly1)−(lx2, ly2), 〈機能〉〔, 〈ファンクションコード〉〔, |B / BF|〕〕

(2)　LINE(cx1, cy1)−(cx2, cy2), 〈文字列〉〔, 〈ファンクションコード〉〔, |B / BF|〕〕

解　説

• (1)の書式は，低解像度グラフィックにより直線や長方形を描きます．〈機能〉は，PSET また PRESET を指定します．PSET で線を描き，PRESET で線を消します．

• (2)の書式は，〈文字列〉の最初の文字を使って直線や長方形を描くものです．

• 最後に，B(Box)を付けると指定した2点を対角線とする長方形の枠を描き，BF(Box Fill)を付けると内側も塗りつぶします．

• 〈ファンクションコード〉はCOLOR文に用いられるものと同じで，カラーモードのときは色を，白黒モードのときはリバースなどの機能を指定することができます．

例

LINE(0, 0)−(100, 50), PSET, , B

• 低解像度グラフィック座標(0, 0)〜(100, 50)を対角線とする長方形を描きます．

プログラム例

```
list↵
100 ' LINE
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 FOR I=1 TO 30
140 LINE(80,50)-(RND(1)*158,RND(1)*90),PSET
150 NEXT I
160 END
Ok
```

• 低解像度グラフィック座標の(80, 50)を始点としていろいろな長さの直線を描きます．

• 140行で，(80, 50)から，乱数で求めた座標までの直線を描きます．

POINT

ポイント：point

機能 指定した低解像度グラフィック座標にドットが描かれているかいないかを返す

書式 POINT(lx, ly)

解説 • 指定した低解像度グラフィック座標にドットが描かれている場合には−1，消去されていたりキャラクタが表示されている場合には0を返します．

例 POINT(10, 10) 返される値 ⇒ 0（ドットがない）

−1（ドットがある）

プログラム例

```
list
100 ' POINT
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 PSET(157,50)
140 X=0:Y=50
150 IF POINT(X,Y)=-1 THEN END
160 PSET(X,Y)
170 X=X+1
180 GOTO 150
190 END
Ok
```

• 画面中央で，横に直線を描きます．

• 130行で，画面の右端中央にドットを描いておきます．

• 140行から180行で，(0, 50)を始点として右にドットを描いていきます．そしてXが157になったとき，150行のIF文の論理式が真になって実行を終了します．

PRESET

ピー・リセット：point reset

機能 低解像度グラフィック座標上のドットを消去する

書式 PRESET (lx , ly〔 ,〈ファンクションコード〉〕)

解説
- 低解像度グラフィックにおいて任意のドットを消去します．
- ここでの〈ファンクションコード〉は意味を持たず，通常指定しません．

例 PRESET(100 , 50)

- 低解像度グラフィック座標(100，50)のドットを消去します．

プログラム例

```
list
100 ' PRESET
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 FOR I=1 TO 1000
140 X=RND(1)*158:Y=RND(1)*96
150 C=RND(1)*7+1
160 PSET(X,Y,C)
170 FOR W=0 TO 99:NEXT W
180 PRESET(X,Y)
190 NEXT I
200 END
Ok
```

- 画面のいろいろな位置にドットを描き，それを消去します．これを1000回繰り返します．
- 140行，150行で乱数を使って座標とカラーナンバを求め，160行でドットを描きます．
- 170行で少し間をおいてから，180行でそのドットを消去します．

参照：PSET

PSET

ピー・セット：point set

機能 低解像度グラフィック座標上のドットを描く

書式 PSET(lx, ly〔,〈ファンクションコード〉〕)

解説
• 低解像度グラフィック座標(lx, ly)にドットを描きます.
• 〈ファンクションコード〉はCOLOR文で用いる場合と同じで，カラーモードのときは色を，白黒モードのときはリバースなどの機能を指定することができます.

例 PSET(100,50,7)
• 低解像度グラフィック座標(100, 50)にファンクションコード7でドットを描きます.

プログラム例

```
list
100 ' PSET
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 FOR I=1 TO 100
140 X=RND(1)*158:Y=RND(1)*96
150 C=RND(1)*7+1
160 PSET(X,Y,C)
170 NEXT I
180 END
Ok
```

• 画面のいろいろな位置にドットを100個描きます.
• 140行，150行で乱数を使って座標とカラーナンバを求め，160行でドットを描きます.

参照：PRESET, COLOR

PUT@

プット・アットマーク：put @

機能 GET@文で配列へ読み込んだキャラクタやグラフィックパターンを表示する

書式 (1) PUT@(cx1 , cy1 | lx1 , ly1) − (cx2 , cy2 | lx2 , ly2) , 〈配列名〉, 〈機能〉

(2) PUT@A(cx1 , cy1) − (cx2 , cy2) , 〈配列名〉

解説 • (1)の書式は，GET@文の書式(1)，(2)で読み込んだデータを表示させる命令です．座標の指定は，キャラクタとして配列に読み込んであるものを表示させる場合はキャラクタ座標で，また，グラフィックパターンとして読み込んである場合はグラフィック座標で行います．

• 〈機能〉は，キャラクタの場合は色，ブリンクなどのファンクションコードを指定し，またグラフィックパターンの場合はPSET，PRESET，OR，AND，NOT，XORを指定します．

• この書式は，GET@A文で読み込まれたデータには使えません．

• (2)の書式は，GET@文の書式(3)で読み込んだものを表示させるものです．座標の指定はキャラクタ座標で行います．

例 PUT@(20 , 20) − (30 , 24) , G% , PSET

• 配列変数G%に入っているグラフィックパターンを，低解像度グラフィック座標(20，20)～(30，24)の範囲に表示します．

プログラム例

```
list
100 ' PUT@
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 COLOR 4,0,1:PRINT CHR$(12);
130 DIM G%(100)
140 LINE(0,50)-(9,59),PSET,B
150 GET@(0,50)-(9,59),G%,G
160 FOR X=20 TO 140 STEP 20
170   PUT@(X,50)-(X+9,59),G%,PSET
180 NEXT X
190 END
Ok
```

• テキスト画面に四角形を8個描きます．

• このプログラムはGET@文と同じものです．

参照：GET@

TERM

ターミナル：terminal

機能 ターミナルモードに入る

書式 TERM〈データのビット長〉，〈パリティ〉，〈ボーレート〉，〈フィードスイッチ〉

解説
• BASICモードからターミナルモードに入ります．

• ターミナルモードでは，RS-232C通信ポートで他のコンピュータや周辺機器との通信が可能となります．

•〈データのビット長〉，〈パリティ〉，〈ボーレート〉，〈フィードスイッチ〉は次のようになっています．

〈データのビット長〉	a：8ビットモード
	j：7ビットモード
〈パリティ〉	0：パリティなし
	1：奇数パリティ
	2：偶数パリティ
〈ボーレートファクタ〉	0：×64
	1：×16
〈フィードスイッチ〉	0：オートL/Fしない
	1：オートL/Fする

•〈データのビット長〉は8ビットモードの場合，最上位ビットでカナと英文字の切り替えを行い，7ビットモードではSI(Shift in)，SO(Shift out)コードで英文字とカナ文字の切り替えが行われます．

•〈パリティ〉はパリティの形式，その有無を指定します．

•〈ボーレートファクタ〉と，本体後部のジャンパスイッチによって実際のボーレートを決定します．ボーレートファクタが×16のとき，後部ジャンパスイッチの値がそのまま採用されます．×64のときはジャンパスイッチの値の4分の1が実際のボーレートとなります．

•〈フィードスイッチ〉の指定を1にすると，CRコードを受信したときにCR(キャリッジリターン)とLF(ラインフィード)を行い改行します．
0のときは，CR，LFコードがそれぞれ独立して働きます．

• ターミナルモードではファンクションキーの6～10が次のような意味を持っています．(　)で囲まれたものは，ターミナルモードに入った直後の状態を表します．

f.6: コントロールコードの表示を行う/(行わない)

f.7: キーボードからの入力を表示する/(表示しない)

f.8: プリンタに出力する/(出力しない)

f.9: 画面の文字をプリンタへ出力する

f.10: プリンタのラインフィードを行う

f.6〜f.8は1度押すと現在の逆の状態になります.

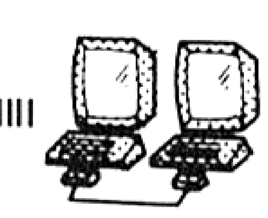
- GRPH + B の入力で，ターミナルモードからBASICに戻ります.

例

TERM j,2,1,1

•データのビット長を7ビットモード，偶数パリティ，ボーレートファクタを×16，オートL/Fを行うターミナルモードに入ります.

実 行 例



RS-232C

```
term j,2,1,1⏎
```

•データのビット長を7ビット，偶数パリティ，ボーレートファクタ16，オートL/Fとしてターミナルモードに入ります.

入出力コマンド

CMD BLOAD

ディスク

シー・エム・ディー・ビー・ロード：cmd binary load

機能　フロッピィディスクから機械語プログラムをロードする

書式　CMD BLOAD〈ファイル名〉〔,〈ロードアドレス〉〕〔,R〕

解説
- 〈ファイル名〉によって指定されたフロッピィディスク上の機械語プログラムファイルをメモリにロードします．このファイルはCMD BSAVEコマンドで作成されたものでなければなりません．
- 〈ロードアドレス〉が省略された場合，CMD BSAVEコマンドで指定したものと同じになります．また，〈ロードアドレス〉が指定されるとその番地からロードされます．
- Rオプションを指定すると，プログラムロード後，セーブの際に指定された〈セーブアドレス〉からプログラムを実行します．このとき既に開かれているファイルはその状態を保持します．〈ロードアドレス〉が指定されている場合は〈ロードアドレス〉からプログラムを実行します．

例　CMD BLOAD "demo.bin", &HD000
- "demo.bin" というファイル名の機械語ファイルをD000H番地からロードします．

実行例

ディスク

```
(a) cmd bload "plydat.bin"↵
    Ok
(b) cmd bload "play.bin",&Hd000,r↵
```

- (a)は，ドライブ1に入っているフロッピィディスクから"plydat.bin"という機械語ファイルをロードします．ロードアドレスは，CMD BSAVEコマンドを実行したときに指定したものと同じになります．
- (b)は，ドライブ1に入っているフロッピィディスクから"play.bin"という機械語ファイルを，D000H番地からロードして，D000H番地から実行を開始します．

参照：CMD BSAVE

CMD BSAVE

ディスク

シー・エム・ディー・ビー・セーブ：cmd binary save

機能 機械語プログラムをフロッピィディスクにセーブする

書式 CMD BSAVE〈ファイル名〉,〈セーブアドレス〉,〈長さ〉

解説
- メモリ上に置かれている機械語プログラムを指定された〈ファイル名〉でフロッピィディスクへセーブします.
- 〈セーブアドレス〉で指定された番地から，〈長さ〉バイトの内容を機械語プログラムとしてセーブします.
- 〈セーブアドレス〉を実行開始番地としておくと，CMD BLOADコマンドにより，機械語プログラムのロードと同時に実行を開始することができます.

例 CMD BSAVE "demo.bin", &HE000, 500
- E000H番地から500バイトの機械語プログラムを "demo.bin" というファイル名でセーブします.

実行例

ディスク

```
cmd bsave "plydat.bin",&hd200,&hd00⏎
Ok
```

- D200H番地からDEFFH番地までの機械語プログラムを，"plydat.bin" というファイル名でドライブ1に入っているフロッピィディスクにセーブします.

参照：CMD BLOAD

CMD CIRCLE

シー・エム・ディー・サークル：cmd circle

機能　円(楕円)を描く

書式

CMD CIRCLE ｜(gx , gy)｜ , 〈半径〉〔 , 〈カラーナンバ〉〕〔 , 〈開始角度〉〕
　　　　　　｜STEP (sx , sy)｜
〔 , 〈終了角度〉〕〔 , 〈比率〉〕

解説

・グラフィック座標(gx, gy)を中心とし，〈半径〉で指定される大きさの円を描きます．

・〈半径〉は，1ドットを単位としたときの長さです．また，〈半径〉が負であった場合は，その絶対値をとります．

・〈カラーナンバ〉を指定すると，指定されたカラーナンバの色で円を描きます．省略された場合，CMD COLOR文で指定されている〈フォアグラウンドカラーナンバ〉で円を描きます．

・〈開始角度〉，〈終了角度〉を指定すると，指定された角度の範囲内のみに円弧を描きます．ただし，角度は真円を基準としたときの値です．したがって，楕円で円弧を描く場合，実際に表示される図形の中心角は，指定した角度と異なることがあります．

・また，〈開始角度〉，〈終了角度〉が負であった場合，その絶対値をとり正にした角度が用いられます．そのとき中心から半径が描かれ，〈開始角度〉，〈終了角度〉共に負であった場合，扇形を描くことができます．

・〈開始角度〉，〈終了角度〉は-2πから$+2\pi$の範囲内にないと "Illegal function call" エラーとなります．

・〈比率〉は，(垂直方向の半径)/(水平方向の半径)で指定します．〈比率〉が省略された場合には，640×200ドットモードでは0.5，320×200ドットモードでは1.0が採用されます．

・320×200ドットモードで1未満の〈比率〉を指定したとき，〈半径〉は水平方向の半径が基準になります．また，1以上の〈比率〉を指定したときには，垂直方向の半径が基準となります．

・640×200ドットモードで1未満の〈比率〉を指定したとき，320×200ドットモードと同様に，〈半径〉は水平方向の半径が基準になります．また，1以上の〈比率〉を指定したとき，〈比率〉が1のときの垂直方向の〈半径〉が基準になります．

・中心の座標はビューポートの外に指定することができます．ただし，極端に大きな円(〈半径〉に16384以上の数値を指定したとき)を表示させた場合，正しく動

作しない場合があります.

例

CMD CIRCLE(100,100),50

• グラフィック座標(100, 100)を中心に，半径50の円を描きます.

プログラム例

```
list
100 ' CMD CIRCLE
110 CMD SCREEN 2,0
120 CMD CLS 3
130 FOR R=1 TO 100
140 CMD CIRCLE(160,100),R,R MOD 3+1
150 NEXT R
160 END
Ok
```

• 中心座標が(160, 100)で，半径が1から100までの同心円を，半径の小さいものから順に描きます.

• 130行から150行のFOR文の制御変数Rを半径として使っています.

• 140行で，中心(160, 100)，半径R，カラーナンバ(R MOD 3)+1の円を描きます. カラーナンバは，Rが変化するにつれて2, 3, 1, 2, 3……と変化していきます.

CMD CLS

シー・エム・ディー・シー・エル・エス：cmd clear screen

機　能　画面を消去する

書　式　CMD CLS〔〈機能〉〕

解　説　•〈機能〉は1，2，3の値をとり，それぞれ次のように働きます．省略されたときは1が選択されます．

1：テキスト画面を消去します．

2：ビューポート内のグラフィック画面を，CMD COLOR文で指定された〈バックグラウンドカラーナンバ〉で消去します．

3：テキスト画面とグラフィック画面の両方を消去します．

•グラフィック画面を消去したとき，LP(Last referenced point)はビューポートの左上の頂点に移動します．また，テキスト画面を消去した場合，カーソルはホームポジション(画面の左上)に移動します．

例　CMD CLS

•テキスト画面を消去します．

プログラム例

```
list↵
100 ' CMD CLS
110 CMD SCREEN 2,0
120 FOR B=3 TO 0 STEP -1
130 CMD COLOR ,B
140 CMD CLS 2
150 NEXT B
160 END
Ok
```

•グラフィック画面を青，緑，赤，黒の順に変化させます．

•130行のCMD COLOR文でバックグラウンドカラーナンバを指定しています．BはFOR文の制御変数なので，3，2，1，0と変化します．

•140行で，130行で指定されたバックグラウンドカラーナンバでグラフィック画面を消去します．

参照：CMD SCREEN, CMD COLOR, CMD VIEW

CMD COLOR

シー・エム・ディー・カラー：cmd color

機能 グラフィック画面のフォアグラウンドカラー，バックグラウンドカラーを指定する

書式 CMD COLOR〔<フォアグラウンドカラーナンバ>〕〔,<バックグラウンドカラーナンバ>〕

解説 •グラフィック画面のフォアグラウンド，バックグラウンドの色をカラーナンバで指定します.

•<フォアグラウンドカラーナンバ>とは，グラフィック画面に点や線を表示するときに使われる色のことです.グラフィック命令(CMD PSET文,CMD LINE文，CMD CIRCLE文など)で<カラーナンバ>の指定を省略した場合，この<フォアグラウンドカラーナンバ>の色が採用されます.

•<バックグラウンドカラーナンバ>とは，グラフィック画面の地の色のことで，CMD COLOR文実行後，CMD CLS文によってグラフィック画面を消去すると，指定したカラーナンバの色によって画面が塗りかえられます．また，CMD PRESET文を<カラーナンバ>の指定なしで実行すると，<バックグラウンドカラーナンバ>の色が採用されます.

•次にカラーナンバと色の関係を示します.

グラフィックモード／カラーナンバ	モノクロモード	アトリビュートカラーモード	4色カラーモード0	4色カラーモード1
0	黒	選択色(黒)	黒	青
1	選択色(白)	テキスト画面の色に依存します.	赤	紫
2	―――	―――	緑	水色
3	―――	―――	選択色(青)	選択色(黒)

•選択色とは，CMD SCREEN文で指定する<選択色のカラーコード>のことで，(　)の色は，CMD SCREEN文で<選択色のカラーコード>が省略されたときのものです．これらの指定の仕方は，CMD SCREEN文の項を参照してください.

•<カラーナンバ>は，モノクロモード，アトリビュートカラーモードのときは0あるいは1を指定していますが，0以外の数値が指定されるとすべて1とみなします．また，4色カラーモード0，1のとき0～3までの値を指定しますが，その

範囲外の数値は４で割った余りが採用されます．

例　CMD COLOR 1,0

・フォアグラウンドカラーに１，バックグラウンドカラーに０を指定します．

プログラム例

```
list
100 ' CMD COLOR
110 WIDTH 40,25:CONSOLE 0,25,1,1
120 COLOR 7
130 CMD SCREEN 2,0
140 FOR F=0 TO 3
150 FOR B=0 TO 3
160 CMD COLOR F,B
170 CMD CLS 3
180 LOCATE 11,12
190 PRINT "cmd color";F;",";B
200 CMD CIRCLE(150,100),70
210 NEXT B
220 NEXT F
230 END
Ok
```

・グラフィックモードを４色カラーモード０にして，フォアグラウンドカラーとバックグラウンドカラーのすべての組み合わせを行います．

・140行のFOR文の制御変数Fは，フォアグラウンドカラーナンバとして用い，150行のFOR文の制御変数Bはバックグラウンドカラーナンバとして用います．

・160行でフォアグラウンドカラーとバックグラウンドカラーを指定します．170行で，指定されたバックグラウンドカラーでグラフィック画面を消去し，200行では，指定されたフォアグラウンドカラーで円を描きます．

参照：CMD SCREEN

CMD COLOR@

シー・エム・ディー・カラー・アットマーク：cmd color @

機　能　テキスト画面に書かれた文字に色や機能を設定する

書　式　CMD COLOR@(cx1，cy1)−(cx2，cy2)，〈ファンクションコード〉

解　説
- テキスト画面のキャラクタ座標の2点(cx1, cy1)，(cx2, cy2)を対角とする四角形の領域に書かれている文字に色を付けたり，ブリンクなどの機能を設定したりします．
- 指定する座標はキャラクタ座標で，次の範囲となります．

　cx1, cx2＝0〜(〈桁数〉−1)
　cy1, cy2＝0〜(〈行数〉−1)

〈桁数〉，〈行数〉はWIDTH文で指定された値です．

- 〈ファンクションコード〉は，CONSOLE文の〈カラー/白黒スイッチ〉で設定されているモードによって持つ意味が異なります．この機能及び指定の仕方は，資料2.3.2のCOLOR文の〈ファンクションコード〉の場合と全く同様ですから，そちらを参照してください．

例　CMD COLOR@(0，0)−(9，9)，4　(カラーモードの場合)
- テキスト画面の左上(10桁×10桁)を緑に設定します．

プログラム例

```
list
100 ' CMD COLOR@
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 CMD SCREEN 1:COLOR 7:CMD CLS 3
130 FOR X=0 TO 639 STEP 4
140 CMD LINE(320,0)-(X,190)
150 NEXT X
160 FOR X=0 TO 7
170 CMD COLOR@(X*10,0)-(X*10+9,23),X
180 NEXT X
190 END
Ok
```

- グラフィック画面に線を描いて三角形をつくり，それを8色に色分けします．
- 120行のCMD SCREEN文でアトリビュートカラーモードを選択しています．
- 160行から180行のFOR文で，画面を縦に8分割して，左から順にカラーナンバを0，1，2，3，4，5，6，7に設定します．

CMD COPY

シー・エム・ディー・コピー：cmd copy

機　能　画面に表示されている情報をプリンタに出力する

書　式　CMD COPY〔〈機能〉〕

解　説

• テキスト画面，グラフィック画面に表示されている文字や図形をプリンタに出力します．

• 〈機能〉は1から5までの値をとり，次のように働きます．省略されたときは3が選択されます．

1：テキスト画面のみをプリンタにコピーします．

2：グラフィック画面のみをプリンタにコピーします．

3：テキスト画面，グラフィック画面の両方をプリンタにコピーします．

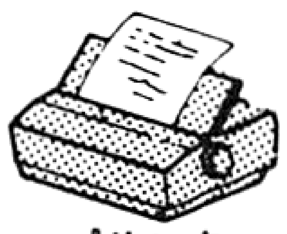
4：グラフィック画面のみをプリンタに，縦方向を2分の1に縮小してコピーします．

5：テキスト画面，グラフィック画面の両方をプリンタに，縦方向を2分の1に縮小してコピーします．

例　CMD COPY 2

• グラフィック画面のみをプリンタに出力します．

プログラム例


プリンタ

```
list
100 ' CMD COPY
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 CMD SCREEN 2,0:CMD CLS 3
130 FOR I=1 TO 30
140 LOCATE RND(1)*70,RND(1)*23
150 PRINT "CMD COPY"
160 CMD LINE(RND(1)*310,RND(1)*190)-STEP(10,10),,B
170 NEXT I
180 FOR I=1 TO 5
190 CMD COPY I
200 NEXT I
210 END
Ok
```

• テキスト画面のいろいろな位置に "CMD COPY" を出力し，グラフィック画面のいろいろな位置に四角形を描いてから，画面をプリンタに出力します．

• 130行から170行までのFOR文で，テキスト画面に "CMD COPY" をいろいろな位置に出力し，グラフィック画面のいろいろな位置に四角形を描きます．

• 180行から200行でCMD COPY文の機能を1から5まで変化させて実行します．

注意： • CMD COPYが使用可能なプリンタは次のとおりです．これら以外のプリンタに対してのCMD COPYの動作は保証されません．

• 縦横の比率がほぼ画面上と同等にコピーできるもの

PC-8023，PC-8023C，PC-8024，PC-8027，PC-8824，PC-8825，PC-PR101，PC-PR201およびこれらの同等品

• 縦長(約4/3倍)になるがコピーできるもの

PC-8821，PC-8822，PC-PR103，PC-PR202，PC-PR403およびこれらの同等品

• グラフィック画面のコピーにおいて，画面に表示されたものと，プリンタにコピーされたものとでは，いくらかイメージが異なります．たとえば，画面上は真円であっても，プリンタに出力した場合に多少縮んだ形になることがあります．

CMD GET@

シー・エム・ディー・ゲット・アットマーク：cmd get @

機能 グラフィック画面のグラフィックパターンを配列に読み込む

書式 CMD GET〔@〕(gx1, gy1)－ |(gx2, gy2) / STEP(sx, sy)| ,〈配列変数名〉〔(〈要素ナンバ〉)〕

解説

• 画面上のグラフィック座標の2点(gx1, gy1)と(gx2, gy2)を対角とする四角形の領域を，〈配列変数名〉で指定された配列変数に読み込みます．指定された座標がビューポートの範囲内にないときはエラーとなります．

•〈配列変数名〉は，グラフィックパターンを読み込む整数型の配列変数の名前です．

•〈要素ナンバ〉は，グラフィックパターンを配列変数に読み込むときに，どの要素から格納し始めるかの指定です．1つの配列変数に対して複数のグラフィックデータを格納する場合に使います．〈要素ナンバ〉を省略したときは配列の最初(〈要素ナンバ〉は0)から格納します．

• CMD GET@文に先だって，この配列変数に必要な大きさをDIM文で確保しなければなりません．この配列変数の大きさは，読み込む領域の大きさ，グラフィックモードによって異なります．

• 1つの配列に1つのパターンしか読み込まない場合は，次の計算式で求めることができます．複数のパターンの場合は，それぞれのパターンについて計算し，その分確保しなければなりません．

〈必要なバイト数〉＝((〈横のドット数〉＊M＋7)¥8)
＊〈縦のドット数〉＋4

M＝1――モノクロモード，アトリビュートカラーモードのとき
M＝2――4色カラーモード0，1のとき

〈添字の値〉＝〈必要なバイト数〉¥2

• このCMD GET@文は，後述するCMD PUT@文と密接な関係にあり，通常ペアで使用されます．

• CMD GET@文の実行後，LPは(gx2, gy2)に移動します．

例 CMD GET@(0,0)－(15,15),G%

• グラフィック座標(0, 0)～(15, 15)の範囲にあるグラフィックパターンを配列変数G%に読み込みます．

プログラム例

```
list⏎
100 ' CMD GET@
110 CMD SCREEN 2,0
120 CMD CLS 3
130 CMD LINE(0,0)-(9,9),2,B
140 CMD LINE(1,1)-(8,8),1
150 CMD LINE(1,8)-(8,1),1
160 DIM G%(16)
170 CMD GET@(0,0)-(9,9),G%
180 FOR I=1 TO 100
190 CMD PUT@(RND(1)*310,RND(1)*190),G%,PSET
200 NEXT I
Ok
```

• グラフィック画面の左上に☒を描き，それと同じ図形をいろいろな位置に100個描きます.

• 130行から150行で，画面の左上に☒を描きます.

• 170行で，描いた図形のパターンを配列G%に読み込みます.

• 180行から200行で，G%に読み込んだグラフィックパターンをいろいろな位置に出力します.

注意：• 配列変数は整数型の一次元配列でなければなりません.

参照：CMD PUT@

CMD INIT

ディスク

シー・エム・ディー・イニット：cmd initialize

機能 フロッピィディスクユニットを初期化する

書式 CMD INIT〔〈機能〉〕

解説
• N_{80}-BASIC DISKバージョンを使用している途中でウォームスタートした後，フロッピィディスクユニットを初期化するために用います．

•〈機能〉は0から2までの値をとり，省略された場合は0と解釈されます．システムディスクとディスクユニットとの組み合わせによって以下のように使い分けなければなりません．

0：最も一般的なディスクユニットの初期化を行います．システムディスクとディスクユニットの関係が次のような場合です．

PC-8087	PC-8881，PC-8881(1)およびその同等品
PC-8037-1W	PC-8031-1Wおよびその同等品
PC-8037-2W	PC-8031-2W，PC-80S31およびその同等品

1：ディスクユニットを片面モードに初期化します．特に〈機能〉を1にする必要があるのは，次の組み合わせのように両面用のディスクユニットを片面モードで使用する場合に限られます．

PC-8037-1W	PC-8031-2W，PC-80S31およびその同等品

片面用のディスクユニットを片面用のシステムディスクで使用している場合は，CMD INIT 1とCMD INIT 0は同じ動作をします．

2：ディスクユニットを両面モードに初期化します．ただし，片面用のディスクユニットを使用している場合は片面モードに初期化します．CMD INIT 2はCMD INIT 0と同じ動作をしますから，通常は使う必要はありません．

例 CMD INIT

• フロッピィディスクユニットの初期化を行います．

実行例 ディスク

```
cmd init
Ok
```

• フロッピィディスクユニットの初期化を行います．

注意：• 両面用のディスクユニットを両面モードで使用しているときにCMD INIT 1は

絶対に実行させないでください．実行させた場合には，ディスクユニットが片面モードになってしまい，N_{80}-BASIC(DISKバージョン)が正しく動作しなくなります．

• N-BASIC(DISKバージョン)の使用中にウォームスタートをした場合，CMD INITコマンドに相当する命令は用意されていません．

CMD LINE

シー・エム・ディー・ライン：cmd line

機能 グラフィック画面に直線や長方形を描く

書式 CMD LINE〔|(gx1, gy1) / STEP(sx1, sy1)|〕−|(gx2, gy2) / STEP(sx2, sy2)|〔,〈カラーナンバ〉〕〔, |B / BF|〕

解説 ・グラフィック座標系の2点(gx1, gy1), (gx2, gy2)を結ぶ直線を描きます．(gx1, gy1)が省略された場合，LPの値をとります．STEPを付けると相対座標による指定となります．

・線の色は〈カラーナンバ〉により指定し，もしこれが省略された場合，CMD COLOR文によって設定されているグラフィック画面の〈フォアグラウンドカラーナンバ〉が採用されます．

・次のパラメータにはBかBFのどちらかを指定することができます．BはBoxの意味で，指定された2点を対角とする長方形を描き，BFはBox Fillの意味で，その長方形を塗りつぶします．

例 CMD LINE(100, 100)−(200, 150), 1, B

・グラフィック座標(100, 100), (200, 150)の2点を対角とする長方形を〈カラーナンバ〉1の色で描きます．

プログラム例

```
list
100 ' CMD LINE
110 CMD SCREEN 2,0
120 CMD CLS 3
130 FOR I=1 TO 100
140 X1=RND(1)*320:Y1=RND(1)*200
150 X2=RND(1)*320:Y2=RND(1)*200
160 C=RND(1)*3+1
170 CMD LINE(X1,Y1)-(X2,Y2),C
180 CMD LINE(X1,Y1)-(X2,Y2),C,B
190 NEXT I
200 END
Ok
```

・対角線が1本入った長方形をいろいろな位置に100個描きます．

・140行から160行で，乱数を使ってグラフィック座標2点とカラーナンバを得ます．そして170行で2点を結ぶ直線を描き，180行でそれを対角線とする長方形を描きます．

参照：CMD COLOR

グラフィックステートメント

CMD PAINT

シー・エム・ディー・ペイント：cmd paint

機能 指定された境界色で囲まれた領域を，指定された色で塗る

書式 CMD PAINT |(gx , gy) | 〔,〈領域色〉〕〔,〈境界色〉〕
| STEP(sx , sy)|

解説 ・(gx, gy)で，塗り始めるグラフィック座標を指定します．

・〈領域色〉，〈境界色〉は共にカラーナンバで指定します．〈領域色〉が省略された場合，CMD COLOR文で指定された〈フォアグラウンドカラーナンバ〉が用いられます．〈境界色〉が省略された場合は，〈領域色〉の指定と同じカラーナンバが用いられます．

・指定された座標が，指定された境界色と同じ色であった場合には，CMD PAINT文は何の画面操作も行いません．

・ビューポートの境界はCMD PAINT文動作の境界とみなされます．また，塗り始めの座標がビューポートの範囲外に指定された場合は，何の動作もしません．

例 CMD PAINT(50,50),0,1

・カラーナンバ1で囲まれた領域を，グラフィック座標(50，50)を基点としてカラーナンバ0の色で塗りつぶします．

プログラム例

```
list
100 ' CMD PAINT
110 CMD SCREEN 2,0
120 CMD CLS 3
130 FOR I=1 TO 100
140 X=RND(1)*320:Y=RND(1)*200
150 C=RND(1)*3+1
160 CMD CIRCLE(X,Y),10,C
170 CMD PAINT(X,Y),C,C
180 NEXT I
190 END
Ok
```

・画面のいろいろな位置に円を描き，その中を塗りつぶします．これを100回繰り返します．

・140行から150行で，円の中心座標と色を，乱数を使って求めます．

・170行で，160行で描いた円の中を，円を描いた線の色で塗りつぶします．

注意：・複雑な図形を塗りつぶすとき，"Out of memory" エラーとなることがあります．

参照：CMD COLOR, CMD VIEW

CMD POINT

シー・エム・ディー・ポイント：cmd point

機能 LP(Last referenced point)を変更する

書式 CMD POINT |(gx , gy) / STEP(sx , sy)|

解説 • N₈₀-BASICは，最後にグラフィック操作の行われた座標を覚えています．この基点のことをLP(Last referenced point)といいます．

• CMD POINT文は，グラフィック操作なしに単にLPを設定する場合に使います．

• STEPを付けると相対座標による指定となります．

例 CMD POINT(10 , 30)

• LPの値を(10 , 30)に設定します．

プログラム例

```
list
100 ' CMD POINT
110 CMD SCREEN 2,0
120 CMD CLS 3
130 FOR I=0 TO 35
140 CMD POINT STEP(8,3)
150 GOSUB 180
160 NEXT I
170 END
180 CMD LINE-STEP(-50,50),1
190 CMD LINE-STEP(100,0),2
200 CMD LINE-STEP(-50,-50),3
210 RETURN
Ok
```

• 画面の左上から右下に向けて三角形を描いていきます．

• 120行のCMD CLS 3でLPは(0, 0)になります．

• 140行のCMD POINT文でLPを少しずつ右下に移動させています．

• 180行から210行は，三角形を描くサブルーチンです．

参照：STATUS POINT

CMD PRESET

シー・エム・ディー・ピー・リセット：cmd point reset

機能 グラフィック画面上の任意のドットを消去する

書式 CMD PRESET {(gx, gy) | STEP(sx, sy)} 〔,〈カラーナンバ〉〕

解説 • 指定したグラフィック座標のドットを消去します．いいかえれば，CMD COLOR文によって設定されている〈バックグラウンドカラーナンバ〉でドットを描きます．

• 〈カラーナンバ〉を指定した場合はCMD PSET文と同じ動作をします．

• 座標値にSTEPを付けた場合は相対座標による指定となります．

• CMD PRESET文の実行後，LPは(gx, gy)に移動します．

例 CMD PRESET(30, 40)

• グラフィック座標(30, 40)のドットを消去します．

プログラム例

```
list
100 ' CMD PRESET
110 CMD SCREEN 2,0
120 CMD COLOR 1,0:CMD CLS 3
130 X=1:Y=1
140 FOR I=1 TO 1600
150 CMD PSET STEP(X,Y)
160 GX=STATUS POINT(0)
170 GY=STATUS POINT(1)
180 IF GX<0 OR GX>319 THEN X=-X
190 IF GY<0 OR GY>199 THEN Y=-Y
200 CMD PRESET STEP(0,0)
210 NEXT I
220 END
Ok
```

• 画面の中を赤い点が移動します．

• 150行で赤い点を描きます．色は120行のCMD COLOR文のフォアグラウンドカラーナンバによって指定されています．

• 180行と190行では，赤い点が画面からはみ出しそうになった場合，移動する方向を逆に変えています．

• 200行で，描いた点を消去します．

参照：CMD COLOR, CMD PSET

CMD PSET

シー・エム・ディー・ピー・セット：cmd point set

機能 グラフィック画面上のドットを描く

書式 CMD PSET |(gx, gy)| 〔,〈カラーナンバ〉〕
|STEP(sx, sy)|

解説
• 指定したグラフィック座標にドットを描きます.
• ドットの色は〈カラーナンバ〉によって指定し，省略された場合は現在の〈フォアグラウンドカラーナンバ〉が用いられます.
• CMD PSET 文実行後，LPは(gx, gy)に設定されます.

例 CMD PSET(30, 40), 1
• グラフィック座標(30, 40)にカラーナンバ1の色でドットを描きます.

プログラム例

```
list
100 ' CMD PSET
110 CMD SCREEN 2,0
120 CMD COLOR 1,0:CMD CLS 3
130 X=1:Y=1
140 FOR I=1 TO 1600
150 CMD PSET STEP(X,Y)
160 GX=STATUS POINT(0)
170 GY=STATUS POINT(1)
180 IF GX<0 OR GX>319 THEN X=-X
190 IF GY<0 OR GY>199 THEN Y=-Y
200 NEXT I
210 END
Ok
```

• 一筆描きをします.
• CMD PRESET 文のプログラム例から CMD PRESET 文を削除したプログラムです. ドットを消していないので，線を描いているように見えます.

参照：CMD COLOR, CMD PRESET

CMD PUT@

シー・エム・ディー・プット・アットマーク：cmd put @

機能 グラフィック画面にグラフィックパターンや漢字を表示する

書式 (1) CMD PUT〔@〕(gx, gy), 〈配列変数名〉〔(〈要素ナンバ〉)〕〔, 〈機能〉〕〔, 〈フォアグラウンドカラーナンバ〉, 〈バックグラウンドカラーナンバ〉〕

(2) CMD PUT〔@〕(gx, gy), KANJI(〈漢字コード〉)〔, 〈機能〉〕〔, 〈フォアグラウンドカラーナンバ〉, 〈バックグラウンドカラーナンバ〉〕

解説
• (1)の書式は，CMD GET@文によって配列に読み込まれたグラフィックパターンをグラフィック画面上の任意の位置に表示します.

• 〈配列変数名〉は表示したいグラフィックパターンが格納されている配列名であり，〈要素ナンバ〉は配列内のどこからデータを取り始めるかの指定です. 〈要素ナンバ〉が省略された場合は，配列の最初(〈要素ナンバ〉は0)から取り始めます. これらの指定は，CMD GET@文で使ったものを指定します.

• 〈機能〉とは，グラフィックパターンを画面に表示する際，いろいろな機能を指定するもので，次に示すものが用意されています.

PSET	配列内のグラフィックパターンをそのまま表示します.
PRESET	モノクロモード，アトリビュートカラーモードの場合は配列内のパターンを反転させて表示します. 4色カラーモードの場合は，各ドットのカラーナンバを3−(そのドットのカラーナンバ)として表示します.
OR	配列内のグラフィックパターンと既に画面上に表示されているグラフィックパターンを1ドットごとにOR(論理和)し，その結果を画面に表示します.
AND	配列内のグラフィックパターンと画面上のパターンをドットごとにAND(論理積)し，その結果を画面上に表示します.
XOR	配列内のパターンをドットごとにXOR(排他的論理和)し，その結果を画面に表示します.

• これらの〈機能〉は画面モードによって演算の対象が違います. モノクロモード，アトリビュートカラーモードの場合，ドットがあるかないかを対象とし，カラーモードの場合，カラーナンバを対象とします. もし，〈機能〉が省略された場合はXORとみなされます.

• 〈フォアグラウンドカラーナンバ〉，〈バックグラウンドカラーナンバ〉は，モノクロモード，アトリビュートカラーモードのときに読み込んだパターンをカラーモードにおいて表示するときにのみ有効なオプションパラメータです. この2つ

のパラメータは両方とも指定するか，両方とも指定しないかのどちらかしか許されません．

•〈フォアグラウンドカラーナンバ〉は，モノクロモード，アトリビュートカラーモードで読み込んだ際にカラーナンバ1に対しての色指定で，これを4色カラーモードで実行すると任意の色に変えることができます．

〈バックグラウンドカラーナンバ〉は同様に黒であったドットに対しての色指定です．これらは共に0〜3のカラーナンバによって指定します．

• CMD GET@文とCMD PUT@文は上記した〈フォアグラウンドカラーナンバ〉，〈バックグラウンドカラーナンバ〉を指定して，モノクロモード，アトリビュートカラーモードで読み込んだパターンを4色カラーモードで表示する用途の他は，原則として同一画面モードで使用するようにしてください．

•(2)の書式は，別売の漢字ROMボードを接続することによって，〈漢字コード〉で指定された漢字などの文字を画面に表示します．

•この書式では，配列内のデータに当たるものがあらかじめ用意されていることを除いて，その使い方および機能は(1)の書式の場合とほぼ同様です．

ただし，漢字パターンは原則として縦16ドット，横16ドットの大きさと決められています(8×8, 8×16のものもあります)．また，漢字パターンはモノクロモードで読み込んだ場合と同じ形で格納されていますから，4色カラーモードにおいて，〈フォアグラウンドカラーナンバ〉，〈バックグラウンドカラーナンバ〉で指定してカラー表示させることも可能です．

例

CMD PUT(10,10),G%,PSET

•グラフィック座標(10, 10)に，配列G%に格納されているデータを表示します．

プログラム例

```
list
100 ' CMD PUT@
110 CMD SCREEN 2,0
120 CMD CLS 3
130 CMD LINE(0,0)-(9,9),2,B
140 CMD LINE(1,1)-(8,8),1
150 CMD LINE(1,8)-(8,1),1
160 DIM G%(16)
170 CMD GET@(0,0)-(9,9),G%
180 FOR I=1 TO 100
190 CMD PUT@(RND(1)*310,RND(1)*190),G%,PSET
200 NEXT I
Ok
```

•グラフィック画面の左上に☒を描き，それと同じ図形をいろいろな位置に100個描きます．

• このプログラムはCMD GET@文と同じものです.

参照：CMD GET@

CMD SCREEN

シー・エム・ディー・スクリーン：cmd screen

機能 グラフィック画面に対して種々のモード設定を行う

書式 CMD SCREEN〔〈グラフィックモード〉〕〔,〈グラフィックスイッチ〉〔,〈選択色のカラーコード〉〕

解説 •〈グラフィックモード〉はグラフィック画面の最も基本的なモードを設定するもので，次の値をとります．

0：モノクロモード(640×200ドット)
1：アトリビュートカラーモード(640×200ドット)
2：4色カラーモード0(320×200ドット)
3：4色カラーモード1(320×200ドット)

•〈グラフィックスイッチ〉を0以外の数字にすると，現在グラフィック画面に描かれているパターンなどは一時的に消去されます．また，0にすればもとに戻ります．

•〈選択色のカラーコード〉はカラーナンバと密接な関係にあり，次のようになります．8色あるカラーコードの中から任意の1色をカラーナンバに対応させることができます．

カラーナンバ ＼ グラフィックモード	モノクロモード	アトリビュートカラーモード	4色カラーモード0	4色カラーモード1
0	黒	選択色(黒)	黒	青
1	選択色(白)	テキスト画面の色に依存します.	赤	紫
2	———	———	緑	水色
3	———	———	選択色(青)	選択色(黒)

•CMD SCREEN文が実行されると，ビューポートは初期化されて，画面のすべてが表示可能となります．

例 CMD SCREEN 2,0,3

•グラフィック画面を4色カラーモード0に設定します．選択色はカラーナンバ3(＝紫)とします．

プログラム例

```
list↵
100 ' CMD SCREEN
110 WIDTH 80,25:CONSOLE 0,25,1,1
120 CMD SCREEN 0
130 CMD CLS 3
140 FOR I=0 TO 199 STEP 3
150 CMD LINE(0,0)-(639,I)
160 NEXT I
170 FOR I=639 TO 0 STEP -8
180 CMD LINE(0,0)-(I,199)
190 NEXT I
200 FOR I=0 TO 3
210 CMD SCREEN I
220 FOR J=1 TO 2000:NEXT J
230 NEXT I
240 END
Ok
```

- モノクロモードの状態で画面に直線を敷きつめ，グラフィックモードを0から3まで変化させます．
- 120行でモノクロモードに設定します．
- 140行から190行で直線を描きます．
- 200行から230行でグラフィックモードを0から3まで変化させます．グラフィックモードが2，3のときは微妙な色の変化が生じます．

CMD VIEW

シー・エム・ディー・ビュー：cmd view

機能 グラフィック画面上での表示領域(ビューポート)を指定する

書式 CMD VIEW(gx1,gy1)―(gx2,gy2)

解説 ・(gx1, gy1), (gx2, gy2)を対角とする長方形の枠内をグラフィック表示領域として指定します. この領域はビューポートと呼ばれます.

・CMD VIEW文は, グラフィックの表示領域の指定を行うだけで, すでに表示されている画面に対する操作は行いません.

・ビューポートの指定を変えるだけで, 1つの図形を描くプログラムでも画面上の異なる位置で描くことができます.

・gx1＜gx2, gy1＜gy2が成り立たない場合, あるいはこれらの座標が絶対座標の範囲から外れている場合には "Illegal function call" エラーとなります.

・一度設定されたビューポートは, 次にCMD VIEW文, あるいはCMD SCREEN文が実行されるまで変化しません.

・LPは, ビューポートの左上の頂点(gx1, gy1)に設定されます.

・システム起動時, CMD SCREEN文実行後のビューポートは次のように設定されています.

モノクロモード, アトリビュートカラーモードのとき

CMD VIEW(0,0)―(639,199)

4色カラーモード0, 1のとき

CMD VIEW(0,0)―(319,199)

・CMD CLS 2または3を実行すると, グラフィック画面はビューポート内のみが消去されます.

例 CMD VIEW(20,20)―(300,70)

・ビューポートを(20, 20)～(300, 70)に設定します.

プログラム例

```
list
100 ' CMD VIEW
110 CMD SCREEN 2:CMD COLOR 2,0
120 FOR I=0 TO 80 STEP 20
130 CMD VIEW(I,I)-(319-I,199-I)
140 CMD CLS 2
150 FOR R=1 TO 200 STEP 5
160 CMD CIRCLE(160,100),R
170 NEXT R
180 NEXT I
190 END
Ok
```

• ビューポートを変化させながら円を描きます.

• 130行でビューポートを設定します. 制御変数 I が大きくなるにつれて, ビューポートはだんだん小さくなります.

参照：CMD SCREEN, CMD CLS.

[1] STATUS POINT

ステータス・ポイント：status point

機能 LP(Last referenced point)の値を返す

書式 STATUS POINT(〈機能〉)

解説 •最後にグラフィック操作の行われた点の位置(LP)を，〈機能〉の指定によりグラフィック座標で返します．

•〈機能〉は0と1の整数値をとり，返す値は次のようになります．

0：X座標を返します．

1：Y座標を返します．

例 STATUS POINT(0) 返される値→ (X座標)

STATUS POINT(1) 返される値→ (Y座標)

プログラム例

```
list↵
100 ' (1) STATUS POINT
110 CMD SCREEN 2,0
120 CMD CLS 2
130 FOR I=1 TO 10
140 CMD PSET(RND(1)*320,RND(1)*200)
150 PRINT "ｸﾞﾗﾌｨｯｸ ｻﾞﾋｮｳ(";STATUS POINT(0);
160 PRINT ",";STATUS POINT(1);")ﾆ ﾃﾝ ｦ ｳﾁﾏｼﾀ｡"
170 NEXT I
180 END
Ok
```

•画面のいろいろな位置にドットを描き，描いた座標を出力します．

•150行と160行のSTATUS POINT文で，140行で描いたドットの座標を求めています．

参照：POINT

(2) STATUS POINT

ステータス・ポイント：status point

機能 指定された座標上に表示されているドットのカラーナンバを返す

書式 STATUS POINT(gx , gy)

解説 • グラフィック座標(gx, gy)で指定された座標上に表示されているドットの色をカラーナンバで返します．指定された座標がビューポートの外にある場合は－1を返します．

例 STATUS POINT(100 , 100) 返される値 (グラフィック座標(100, 100)にあるドットのカラーナンバ)

プログラム例

```
list
100 ' (2) STATUS POINT
110 CMD SCREEN 2,0
120 CMD CLS 2
130 FOR R=1 TO 100
140 CMD CIRCLE(160,100),R,R MOD 4
150 NEXT R
160 FOR I=50 TO 140 STEP 10
170 PRINT "ク゛ラフィック ザ゛ヒョウ(";I;",";I;")";
180 PRINT "ニ アル テン ノ カラーナンバ゛ ハ";
190 PRINT STATUS POINT(I,I);"デ゛ス。"
200 NEXT I
210 END
Ok
```

• 画面に 4 種類の色で同心円を100個描き，いろいろな座標のカラーナンバを出力します．

• 130行から150行で円を描きます．

• 190行で，点(I, I)のカラーナンバを出力します．

STATUS VIEW

ステータス・ビュー：status view

機能 現在のビューポートの設定位置を返す

書式 STATUS VIEW(〈機能〉)

解説
- CMD VIEW文により設定されている現在のビューポートの位置(gx1, gy1)−(gx2, gy2)を返します.
- 〈機能〉は0〜3の整数値です.

  0：ビューポート左上の頂点のX座標(gx1)

  1：ビューポート左上の頂点のY座標(gy1)

  2：ビューポート右下の頂点のX座標(gx2)

  3：ビューポート右下の頂点のY座標(gy2)

例 STATUS VIEW(0) 返される値 ⇒ (ビューポート左上の頂点のX座標値)

プログラム例

```
list
100 ' STATUS VIEW
110 CMD SCREEN 2,0
120 CMD VIEW(10,20)-(30,40)
130 PRINT "ゲンザイ ノ ビューポート ハ ";
140 PRINT "VIEW(";STATUS VIEW(0);",";
150 PRINT STATUS VIEW(1);")-(";
160 PRINT STATUS VIEW(2);",";
170 PRINT STATUS VIEW(3);") デス。"
180 END
Ok
run
ゲンザイ ノ ビューポート ハ VIEW( 10 , 20 )-( 30 , 40 ) デス。
Ok
```

- 現在のビューポートの位置を出力します.
- 140行から170行で，120行で指定したビューポートの位置を出力します.

参照：CMD VIEW

特殊コマンド

FORMAT

ディスク

フォーマット：format

機能 ミニフロッピィディスクを初期化(フォーマット)する

書式 FORMAT〔〈ドライブ番号〉〕

解説 • FORMATコマンドはミニフロッピィディスクの初期化を行います．しかし，FORMATコマンドを実行したフロッピィディスクは，FAT，ディレクトリ，IDの初期化が行われていませんから，N-BASICから使うことはできません．N-BASICから実行できるのは，DSKI$関数，DSKO$文だけです．

• FORMATコマンドは通常使う必要はありません．逆にFORMATコマンドによって重要なデータを消してしまう危険性もありますから，使わない方がよいでしょう．フロッピィディスクを初期化する場合は，N-BASIC DISKバージョン用のシステムディスクに入っている"format"というユーティリティを使用してください．

例 FORMAT 2

• ドライブ2に入っているフロッピィディスクを初期化します．

実 行 例

ディスク

```
format 2↵
Ok
```

• ドライブ2に入っているフロッピィディスクを初期化します．

注意：• FORMATコマンドを実行すると，フロッピィディスクに書き込まれていたファイルはすべて消去されます．したがって，重要なファイルが書き込まれているフロッピィディスクに対してFORMATコマンドを実行しないように注意してください．

• FORMATコマンドが使用できるのはインテリジェントタイプのミニフロッピィディスクだけです．

MOUNT

マウント：mount

機能 ディスクのファイル配置表(FAT)を読んで以後ディスクの入出力を可能にする

書式 MOUNT〔〈ドライブ番号〉〔,〈ドライブ番号〉 ... 〕〕

解説

• フロッピィディスクとの入出力(FILES, LOAD, OPEN, PRINT#)などを行う前にはMOUNTコマンドを実行しなければなりません．

•〈ドライブ番号〉を省略したときは，現在接続されているすべてのドライブが入出力可能となります．この場合，MOUNTコマンドはドライブ番号の大きい方から順番に行われます．

例 MOUNT 1

• ドライブ1を入出力可能とします．

実行例

ディスク

```
mount 1⏎
Ok
```

• ドライブ1を入出力可能にします．

参照：REMOVE

REMOVE

ディスク

リムーブ：remove

機能 フロッピィディスクに配置表(FAT)を書き込んでフロッピィディスクの入出力を終了する

書式 REMOVE〔〈ドライブ番号〉〔,〈ドライブ番号〉...〕〕

解説
- REMOVEコマンドはMOUNTコマンドの逆の動作をします．フロッピィディスクをドライブから取り出す前に必ず実行しなければなりません．
- REMOVEコマンドを実行すると，そのときメモリ上にある配置表(FAT)をフロッピィディスクのディレクトリトラックに書き込みます．

例 REMOVE

- 現在接続されているすべてのドライブの入出力を終了します．

実　行　例

ディスク

```
remove ↵
Ok
```

- 現在接続されているすべてのドライブの入出力を終了します．

注意：• フロッピィディスクの入出力を終了し，ドライブからフロッピィディスクを取り出すときは，必ずREMOVEコマンドを実行してください．
もしREMOVEコマンドを実行しなかった場合，取り出したフロッピィディスクの更新が行われず，また，そのドライブに入れられた次のフロッピィディスクには誤った配置表が書き込まれてしまい，両方のフロッピィディスクの配置表(FAT)が破壊されてしまいます．

参照：MOUNT

資料3　N$_{80SR}$-BASICの予約語

ABS	DEFDBL	INSTR	OUT	STR$
AND	DEFINT	INT	PAINT	STRING$
ASC	DEFSNG	KEY	PEEK	SWAP
ATN	DEFSTR	KILL	PLAY	TAB(
ATTR$	DELETE	LEFT$	POINT	TALK
AUTO	DIM	LEN	POKE	TAN
BCD$	DSKF	LET	PORT	THEN
BEEP	DSKI$	LFILES	POS	TIME$
BGM	DSKO$	LINE	PRESET	TO
BLOAD	ELSE	LIST	PRINT	TROFF
BSAVE	END	LLIST	PSET	TRON
CDBL	EOF	LOAD	PUT	USING
CHR$	EQV	LOC	READ	USR
CINT	ERASE	LOCATE	REM	VAL
CIRCLE	ERL	LOF	RENUM	VARPTR
CLEAR	ERR	LOG	RESTORE	VIEW
CLOAD	ERROR	LPOS	RESUME	VOICE
CLOSE	EXP	LPRINT	RETURN	WAIT
CLS	FIELD	LSET	RIGHT$	WIDTH
CMD	FILES	MERGE	RND	XOR
COLOR	FIX	MID$	ROLL	+
CONSOLE	FN	MKD$	RSET	−
CONT	FOR	MKI$	RUN	*
COPY	FPOS	MKS$	SAVE	/
COS	FRE	MOD	SCREEN	^
CSAVE	GET	MON	SET	'
CSNG	GOSUB	MOTOR	SGN	>
CSRLIN	GOTO	NAME	SIN	=
CVD	HEX$	NEW	SOUND	<
CVI	IF	NEXT	SPACE$	¥
CVS	IMP	NOT	SPC(	
DATA	INIT	OCT$	SQR	
DATE$	INKEY$	ON	STATUS	
DEC	INP	OPEN	STEP	
DEF	INPUT	OR	STOP	

資料4　FM音源とシンセサイザーIC

この節では，音作りに必要なFM音源とシンセサイザーIC(OPN)について解説します．したがって，自分で音色を作りたい方，あるいはOPNのレジスタを知りたい方以外は読む必要はありません．

資料4.1　FM音源

PC-8001MKⅡSRのOPNには49種類もの音色がすでに用意されていますが，ここではOPNを制御してさらに自分で音色を作る方のために，FM音源の原理について解説します．

1　FM音源とは

FM音源はデジタル音源が基礎となっています．デジタル音源は音を何もない状態から“発生”させるのではなく，コンピュータに使われているのと同じような記憶回路から，いろいろなタイミングで音を読み出し“再現”する方式をとっています．

この記憶回路にはサイン波だけが，“1”と“0”だけでデジタル符号化されて入っています．

そして入力信号(読み出すタイミング)により基準となる信号(記憶回路)の形を変えて出力する方式(周波数変調，Frequency Modulation)を使用しているためFM音源と呼ばれています．

以下にデジタル音源の概要を説明します．

2　“波形＋時間”が“音”になる

ここにひとつの記憶回路があるとします．その中にはサイン波が，図4.1のように10個の電圧として蓄えられています．a～jの電圧が四捨五入され，二進化されて入っています．大切なのは最も下側の欄で，サイン波はすべてこのような“1”と“0”に置き換わっているということです．しかしこれは説明のための単なる一例で，実際の楽器では，サイン波は必ずしも10個の電圧に分割されているわけではありません(もっと多くの電圧に分割されている)し，電圧の値もこうとは限りません．

このような形で記憶回路に入っているのは，“サイン波のもと”でしかありません．というのは，記憶回路にはサイン波を構成する電圧が順番どおりに入っているだけで，それらをどういったタイミングで並べるかのデータは入っていないからです．

aとb，bとcの間を，どのくらいの時間にするか(1マイクロ・セカンドなのか1秒なのか……)のデータは全然入っておらず，それが決まらなければ“音”にはなりません．aとb，bとcといった各電圧の間の“時間”を決めて，初めて“音”になります．もしも“時間”が決まらなければサイン波の1サイクルの時間もわからないわけですから，周波数が決まりませ

ん．波形と周期(１サイクルの所要時間)が決まって，初めて“音”になります．

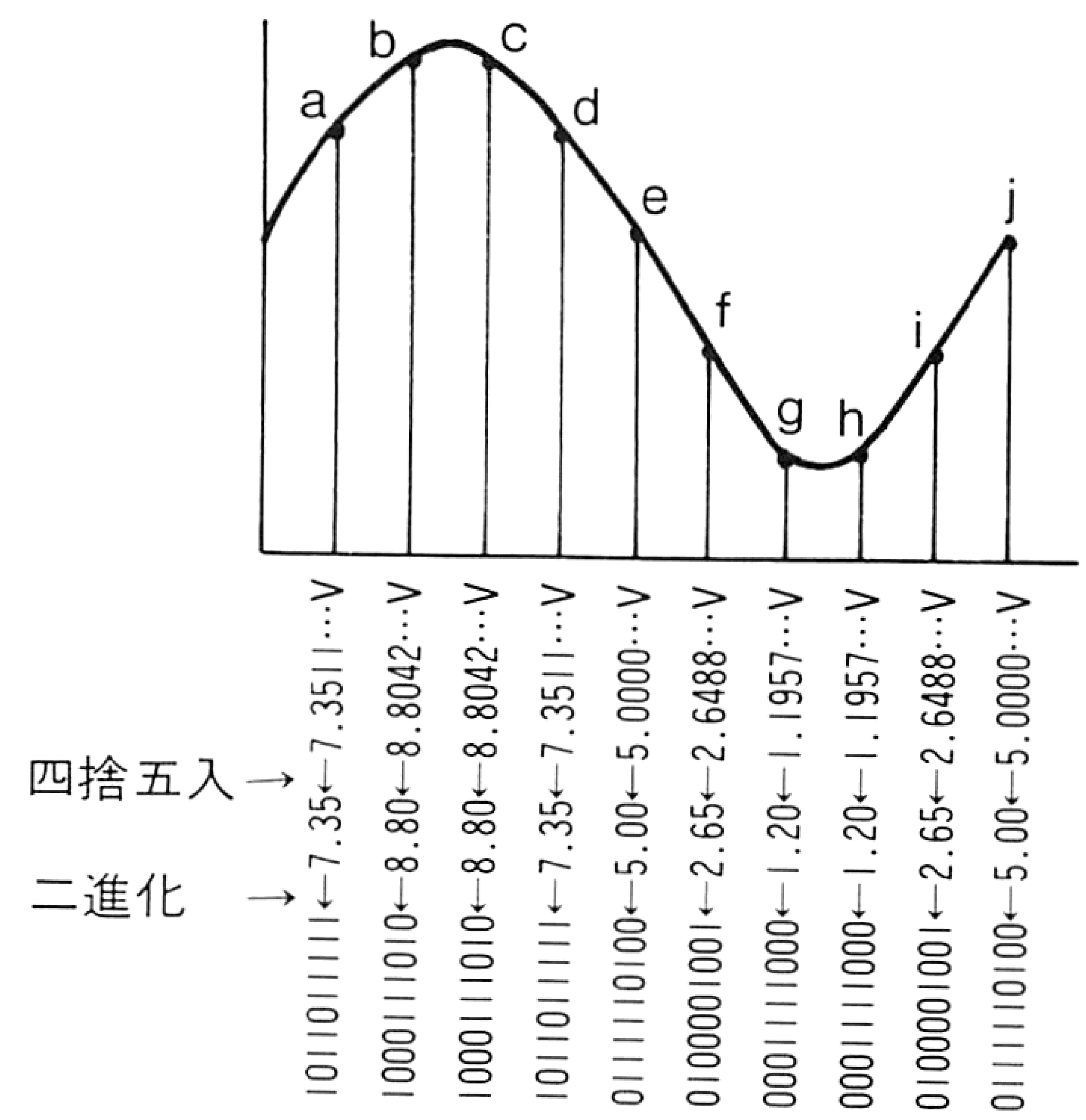


図4.1

３　記憶回路に“時間”を与えると……

記憶回路に“時間”のデータを与えるにはどうしたらいいのでしょうか？ 図4.2はノコギリ波です．今かりに，このノコギリ波が0Vから上昇し，10Vでストンともとに戻るような振幅だったとしてこれを記憶回路に与えます．そして，ノコギリ波の電圧が1Vの時に記憶回路のa.の電圧を，2Vならb. ……としていき，10Vになったらj.の電圧を記憶回路から出力させるようにしておきます．このような約束にしておくと，ノコギリ波の電圧は直線的に上昇しますから，a.とb.，b.とc.といった隣り合った電圧が一定の時間間隔で出力に出てくることになります．図4.3がそのようすです．

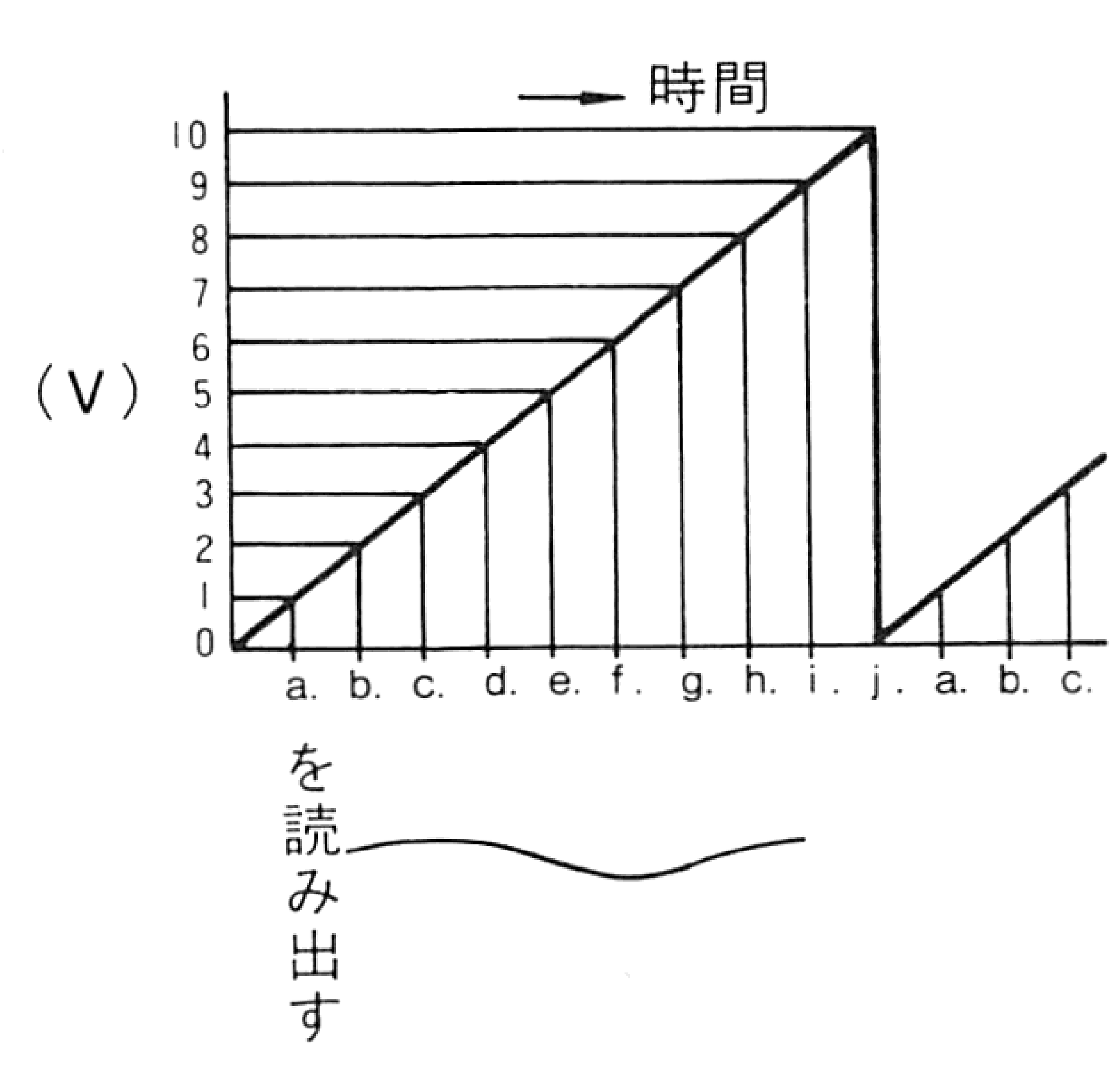


図4.2

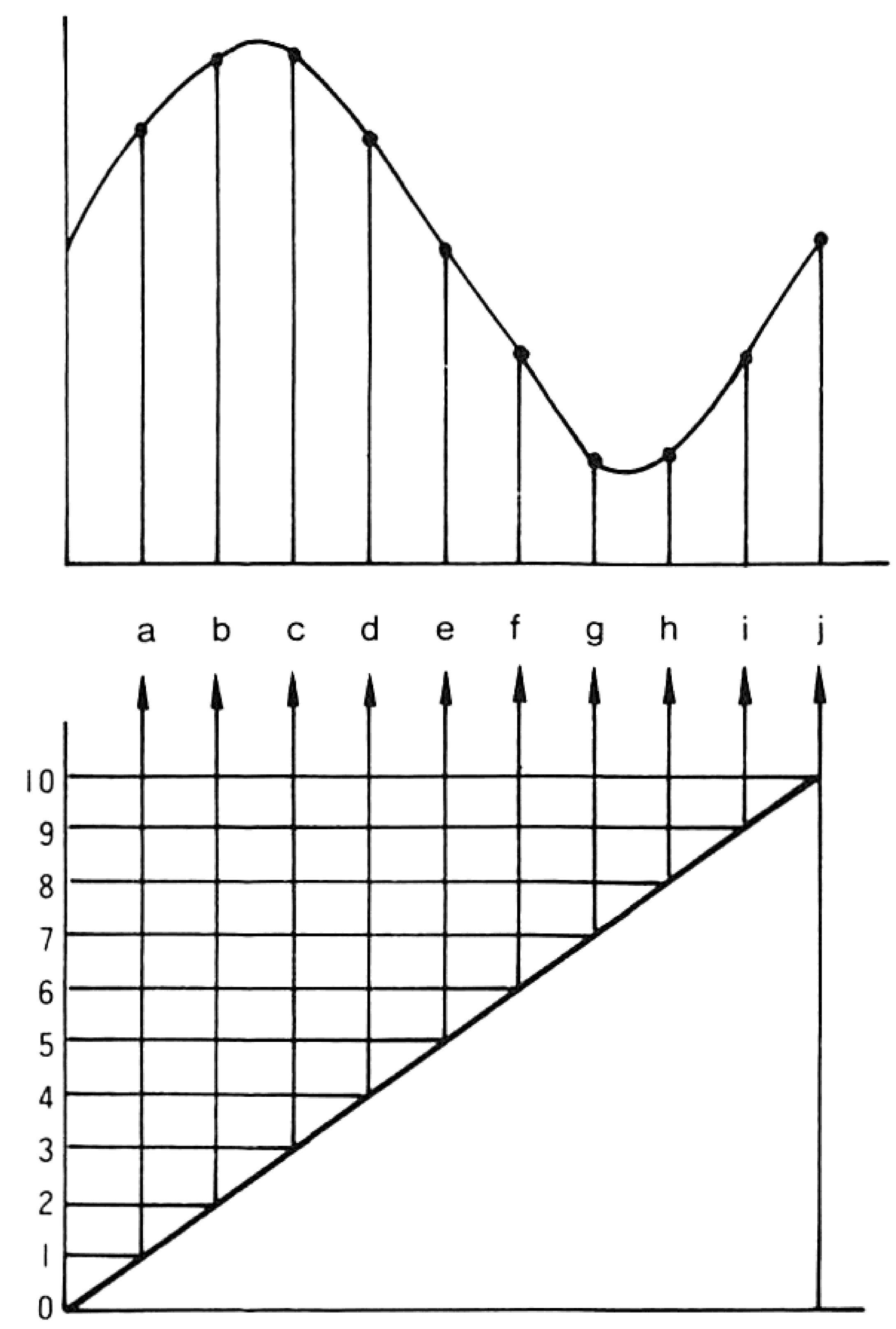


図4.3

この方法によって，記憶回路からきれいなサイン波が出てきます．そして，ここで大切なのは「読み出されたサイン波の周波数は，読み出しに使ったノコギリ波の周波数と同じである」ということです．図4.4を見てわかるように，ノコギリ波の１サイクルとサイン波の１サイクルは常に一致しています．そして，１サイクルに要する時間が同じであれば周波数も同じになります．（１サイクルが１ミリ・セカンドの波形は，これが連続している場合，１秒間に1000回くり返されます．つまり１秒間に1000サイクルですから，これは1000Hzの周波数にあたります.）

これが，“デジタル音源”の基礎です．

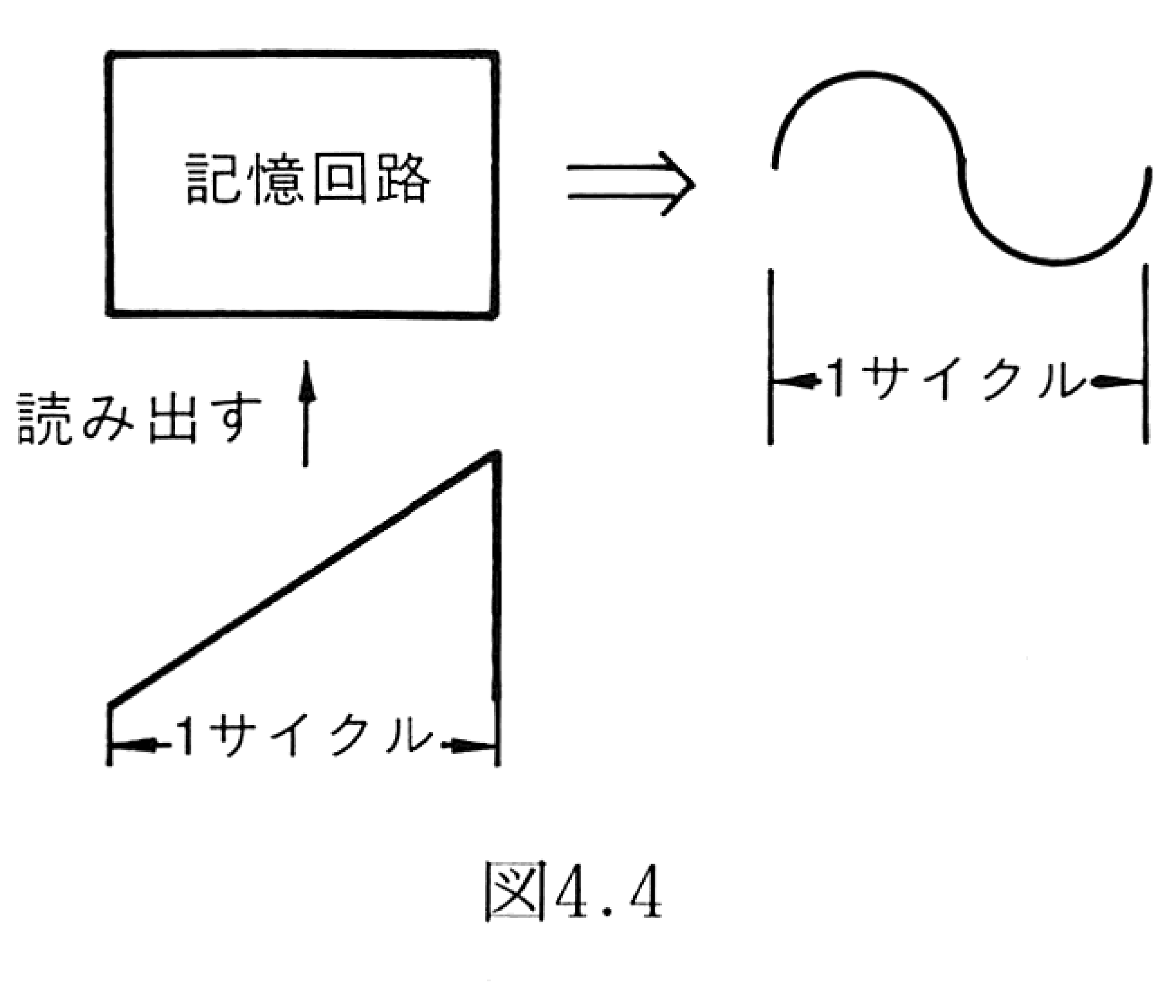


図4.4

4　記憶されているのは“サイン波”だけ

FM音源は，デジタル音源のひとつの発展型と考えられます．ですから，“記憶回路からノコギリ波を使ってサイン波を読み出す”のが第一歩となります．

図4.4をもう一度見てください．ここで，ノコギリ波を図4.5のように細工して記憶回路を読み出してみましょう．出てくるのはひしゃげたサイン波になってしまいます(図4.6)．どうしてでしょうか？ 図4.7を見るとわかるように折れ曲ったノコギリ波は，2つのノコギリ波が合成されたもの，と考えれば簡単になります．

それでは図4.8のように，今度は逆に折ったらどうなるでしょうか？ さっきとは逆にひしゃげたサイン波になります(図4.9)．

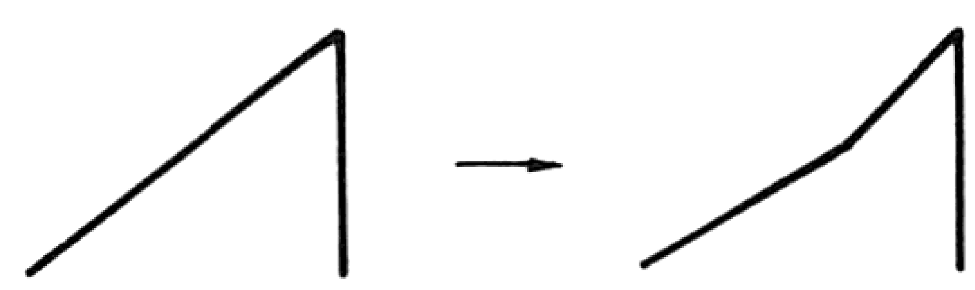


図4.5

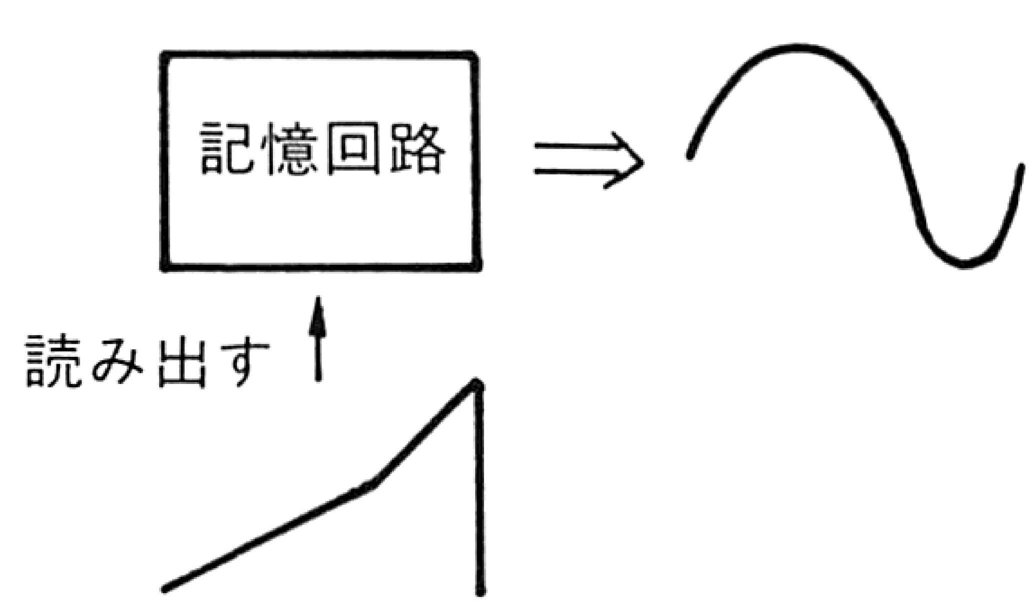


図4.6

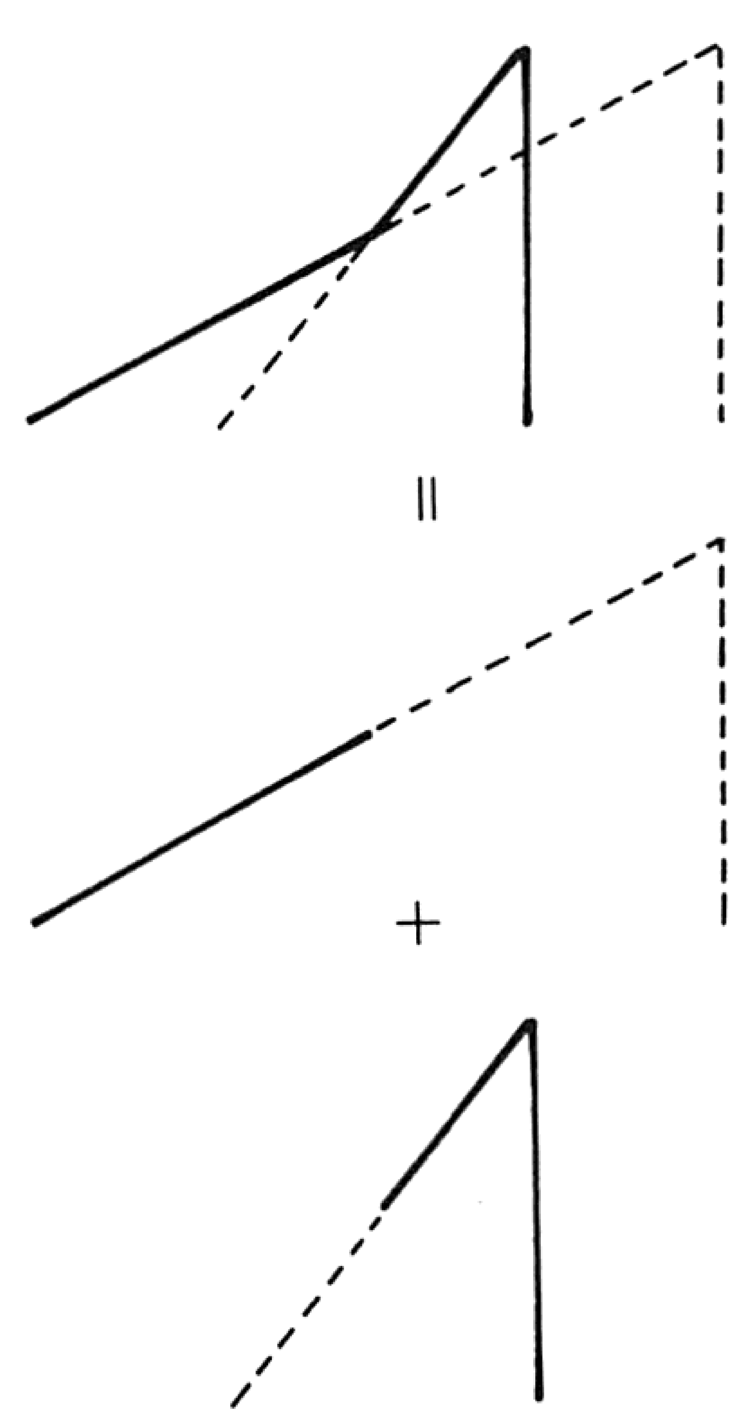


図4.7

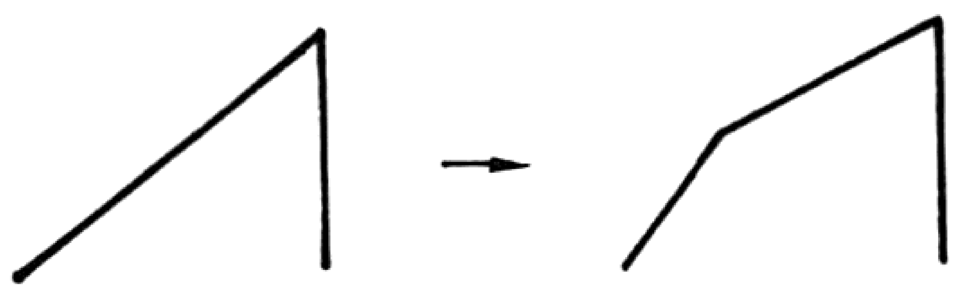


図4.8

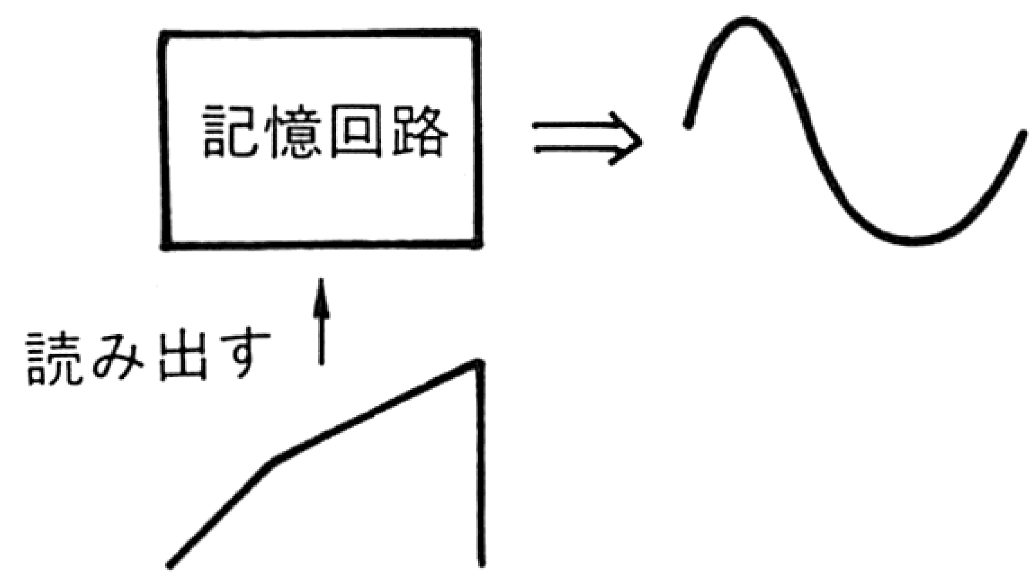


図4.9

読み出しに使うのが，必ずしも正しいノコギリ波でなくてもいいのなら，たとえば図4.10のようなもっと複雑な折れ線の波形で読み出してみましょう．この折れ線で問題なのは㋺～㋩間です．なぜなら㋑～㋺，㋩～㋥間は，図4.11のように考えればそれぞれ“ノコギリ波の一部”と見なせますが，㋺～㋩では傾斜が逆です．つまり，逆向きのノコギリ波になってしまいます(図4.12)．

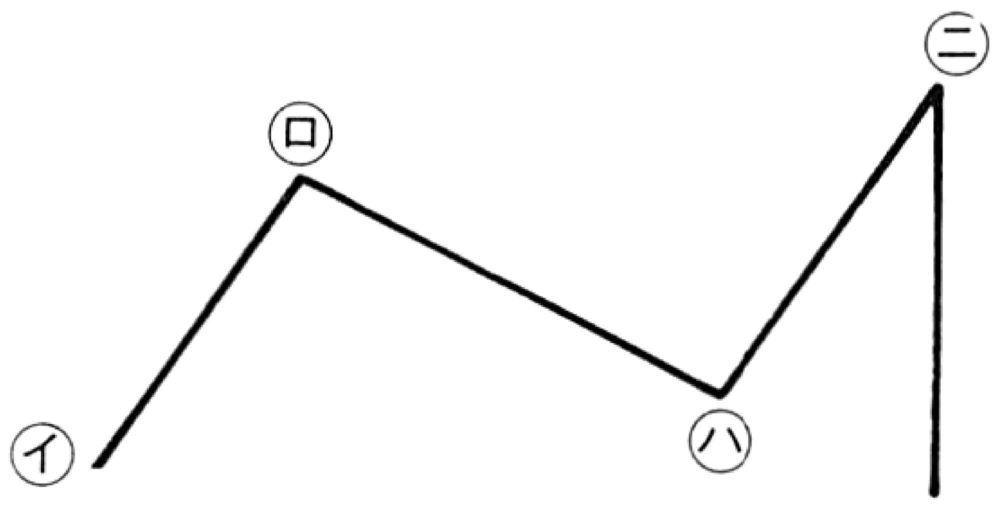


図4.10

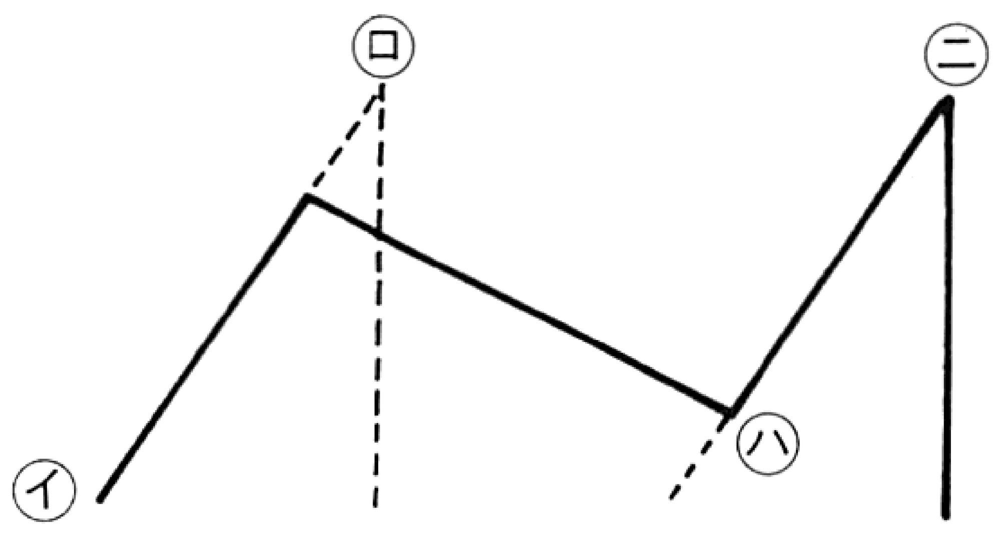


図4.11

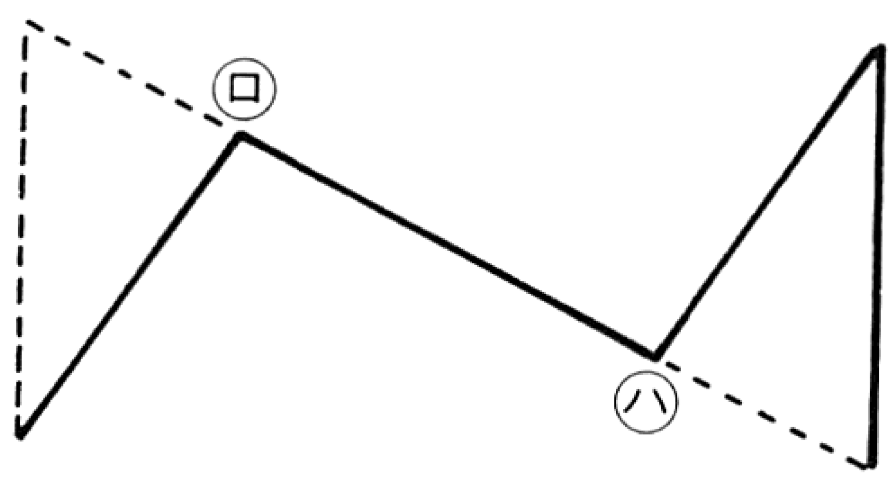


図4.12

そして図4.13が記憶回路から出てくる波形です．これはもうひしゃげすぎていてサイン波とは呼びがたいですね．しかし，これで“逆向きのノコギリ波”でも，読み出しには全く支障がないことがわかりました．どうしてでしょう？

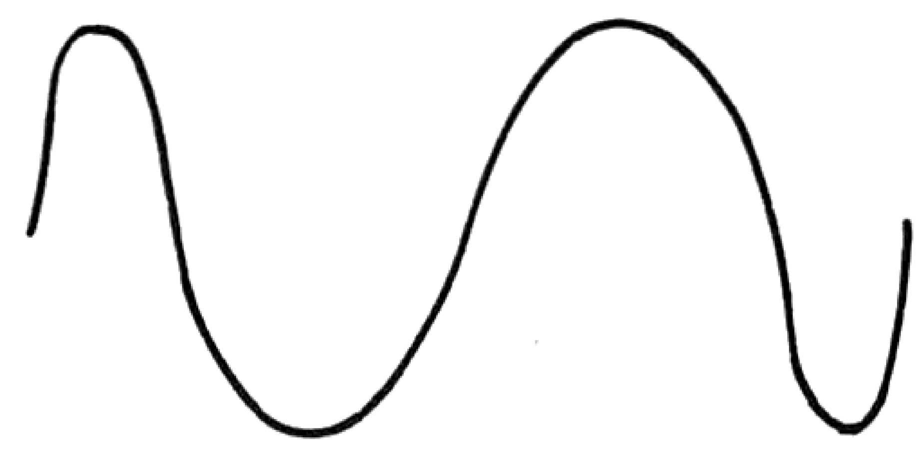
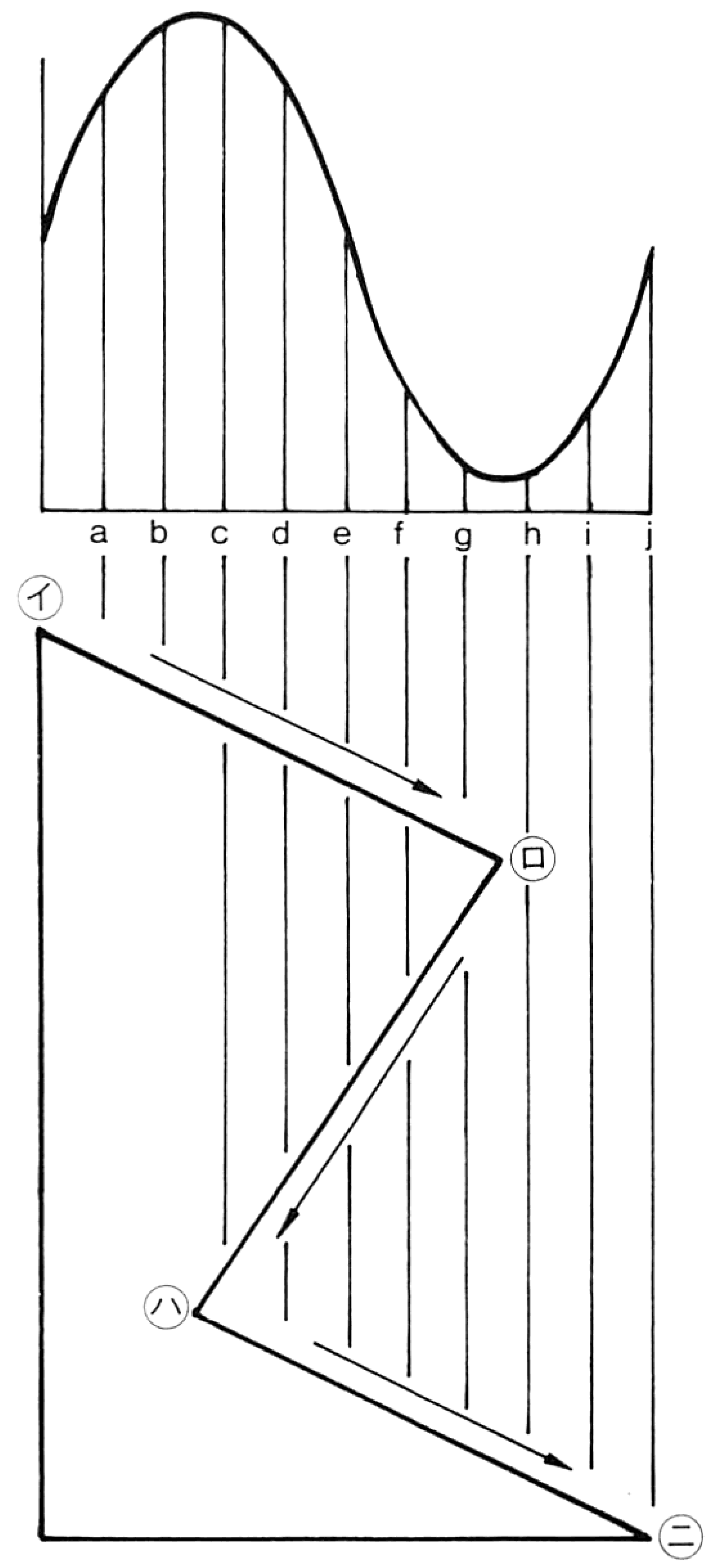


図4.13

図4.14で説明をしましょう．これはただ，読み出し用の波形を90度回転しただけのものです．まず㋑～㋺では，記憶回路の中の電圧を順番に読み出します．ここまでは普通ですが，次に㋺～㋩では，今度は記憶されている電圧を逆の順番で読みます．a. b. c.……と読んでいたものが，逆になるとg. f. e.……になります．そして，㋩～㋥では再び正しい順番に戻ります．



㋺～㋩間は逆読みする

図4.14

このように記憶回路を読み出すのは，どんな波形(ノコギリ波でなくても)でもかまいません．そして，読み出しに使う波形によって，記憶回路に入っているサイン波は，いろんな形に変形されて出力されます．これがFM方式の最も大切な基本です．

5 “ノコギリ波＋サイン波”で読み出す

ところで図4.10をもう一度見てください．一見メチャクチャな波形ですが，実はこれは，コギリ波と三角波が足し合わされたものです．図4.15のように，シンプルな波形どうしでも，足し合わせる(以後“加算”という)ことによって，かなり複雑な波形にすることができます．

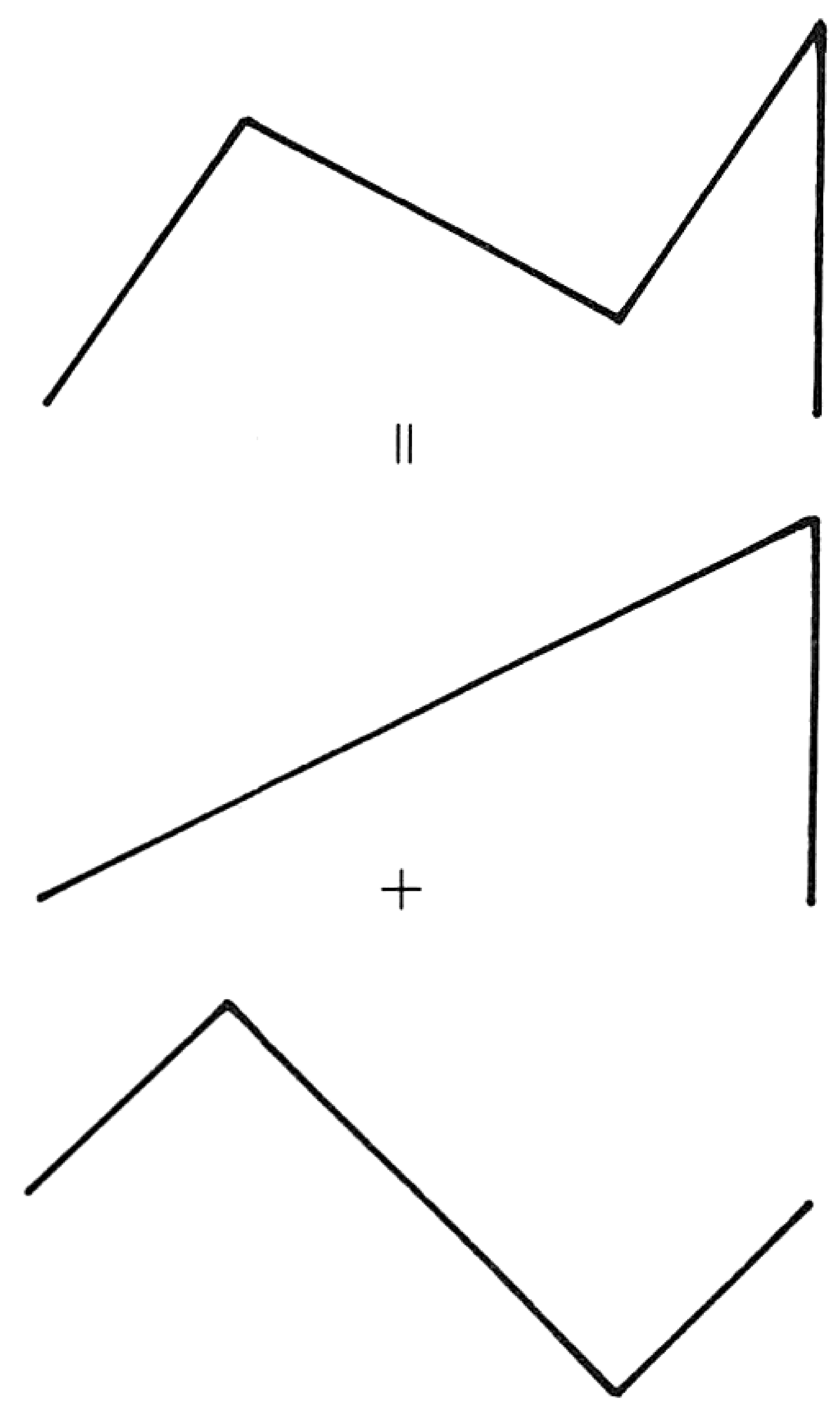


図4.15

三角波の代わりにサイン波をノコギリ波に加算したらどうでしょう．図4.16がそれです．そして，図4.17が記憶回路から出てくる波形です．

図4.16では,ノコギリ波にくらべてサイン波の振幅を小さくしておきました．今度は,サイン波の振幅を大きくしてみましょう．図4.18を見てください．これを使って記憶回路を読み出すと，図4.19の波形が得られます．図4.17と比べてみてください．同じ〔ノコギリ波＋サイン波〕で読み出しても，加算したサイン波の振幅を変えると，読み出される波形は違ったものになってきます．

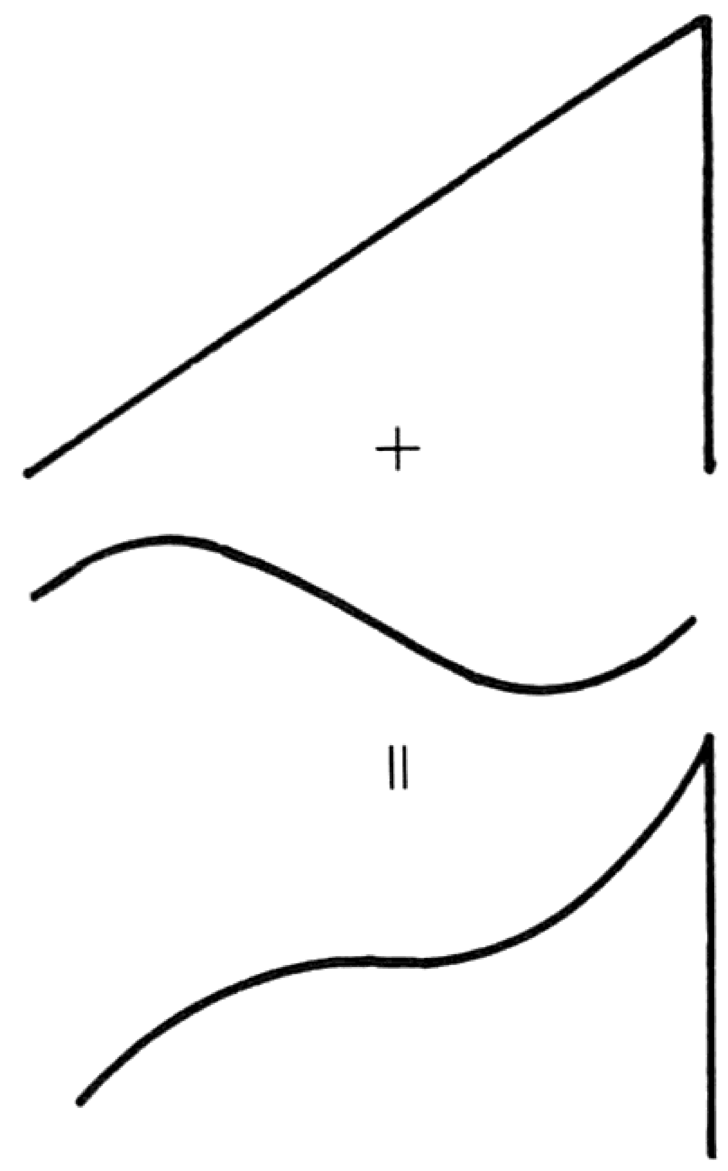


図4.16

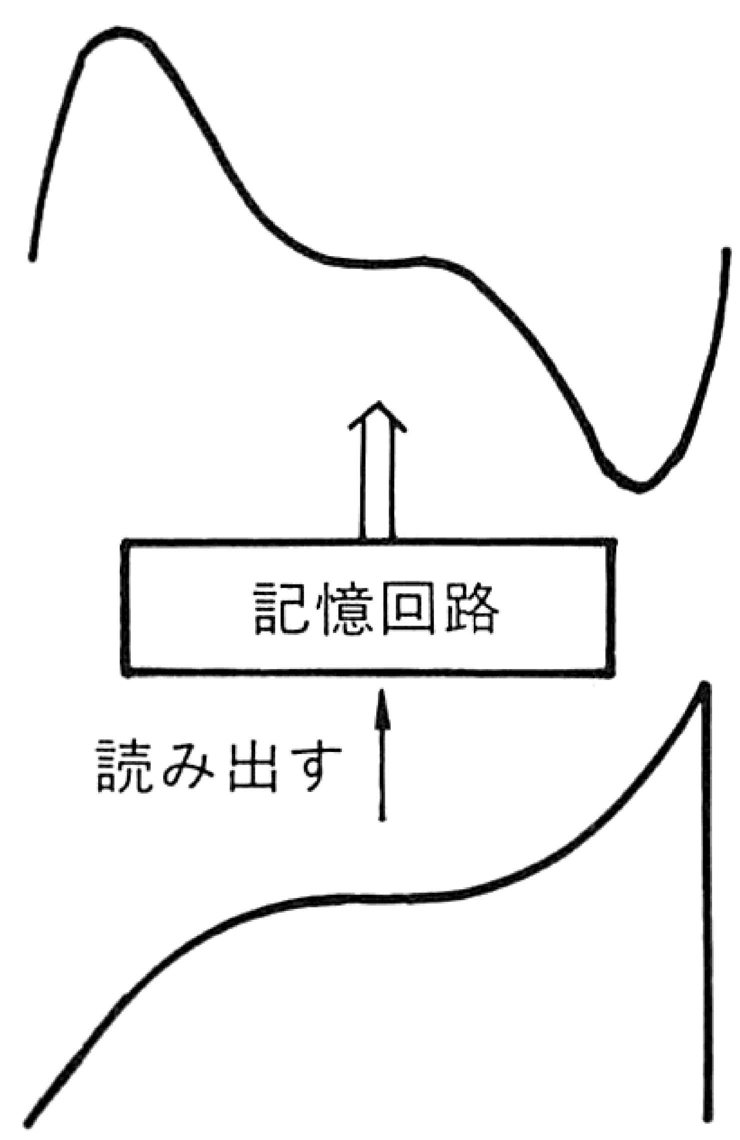


図4.17

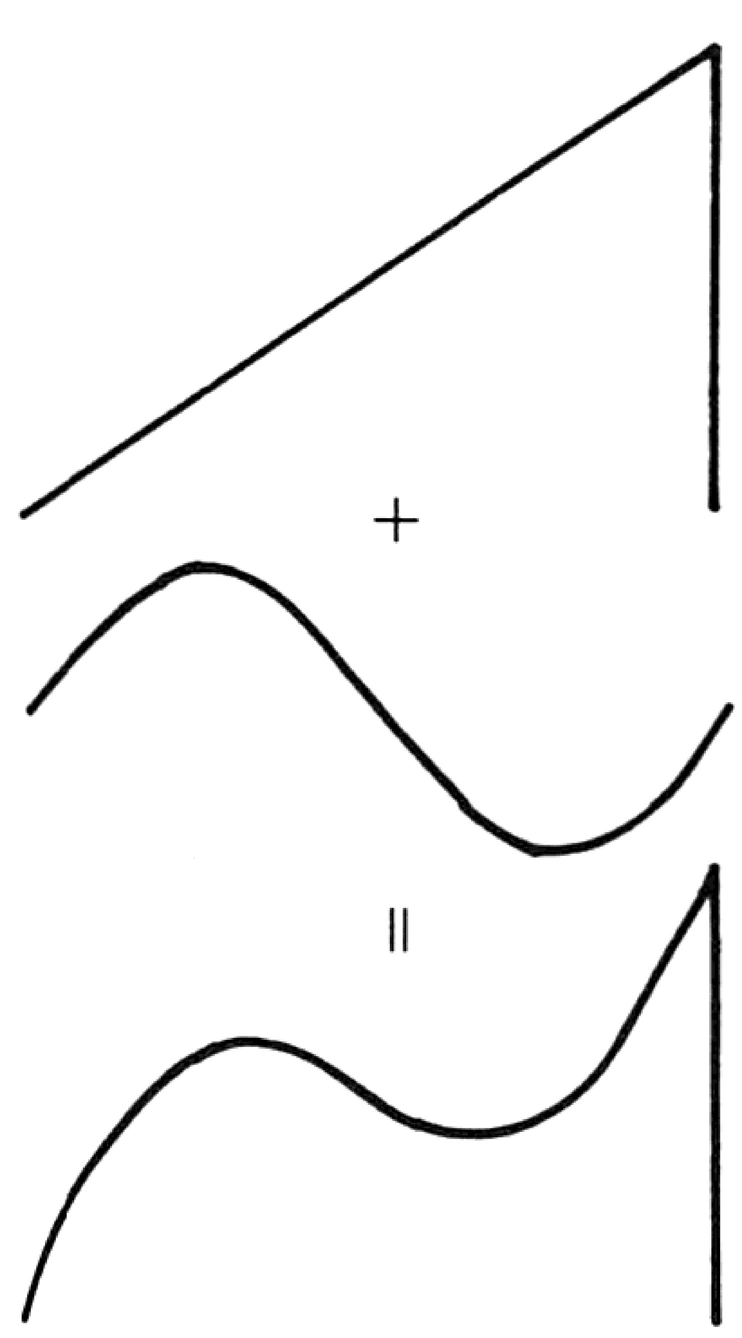


図4.18

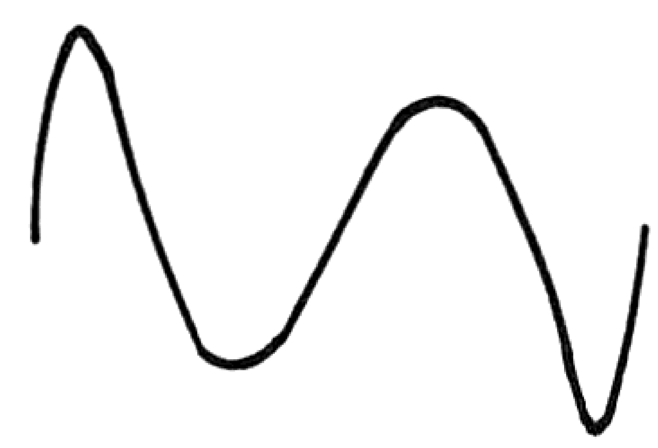


図4.19

6　ノコギリ波が“周波数”，サイン波が“音色”を決める

今まで書いた“読み出し”の例をよく見直してみると，サイン波の振幅を変えたり，あるいはサイン波の代わりに三角波を使った場合でも，読み出された波形の１サイクルの時間はすべてノコギリ波によって決定されています．ノコギリ波にどんな波形を加算しようとも，それによって読み出された波形の１サイクルは，ノコギリ波の１サイクルの時間と一致しています．これはデジタル方式の場合と全く同じで，ノコギリ波がすべての周波数を決めていることになります．

一方，ノコギリ波に加算されたサイン波はどんな役割りを持っているのでしょうか？ これらは，記憶回路から出てくる波形自体を決めています．つまり，音色を決定しています．

このようにノコギリ波とサイン波は，それぞれ決定すべき大きな役割りを持っています．

7　2つの波形の周波数を変えると

これまでの例では，ノコギリ波にサイン波を加算するとき，両者の周波数比は1：1，つまり同じ周波数でした．そこで今度は，周波数比を変えて考えてみましょう．ノコギリ波は，出力の周波数を決定するものですからその周波数は変えません．変えるのはサイン波の周波数です．

まず，ノコギリ波の周波数に対して，サイン波の周波数を２倍にとってみましょう．周波数比は1：2です．図4.20がその波形です．

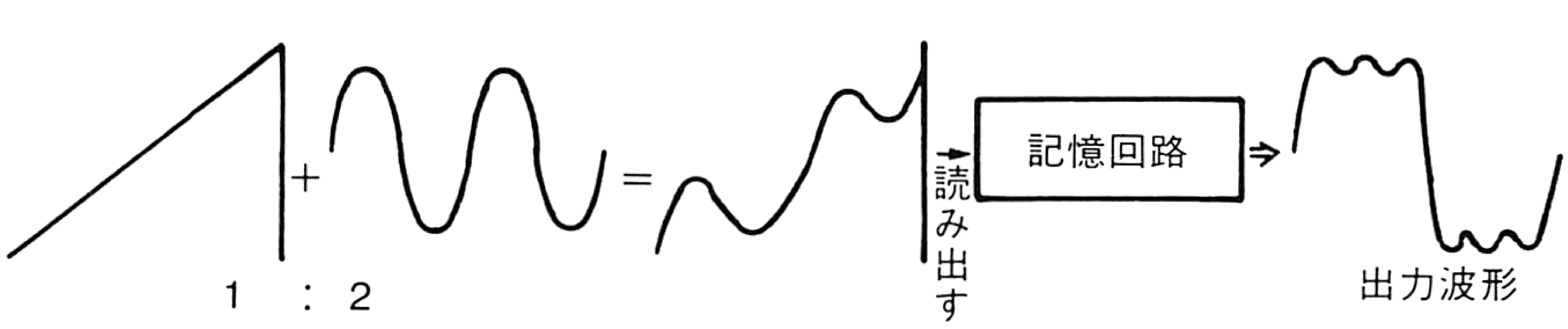


図4.20

次に周波数比を1：5にしてみましょう(図4.21)．1：2のときにくらべて，ますます複雑な波形になります．

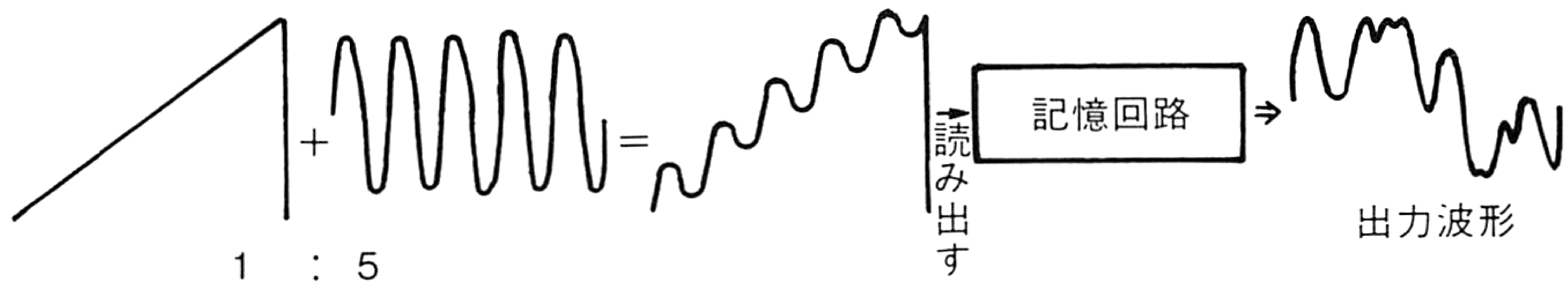


図4.21

このようにノコギリ波に同じ振幅のサイン波を加算した場合でも，サイン波の周波数を変えていくと記憶回路から読み出される波形も大きく変化します．このようにノコギリ波：サイン波の周波数比は，自由に変えられます．また，この比は整数比でなくてもよく，たとえば，1：8.265のような半端な数字でもかまいません．例として図4.22に，1：2.5，1：3，1：4の場合に読み出された波形を載せておきます．これらは，加算したサイン波の振幅を一定にして周波数比だけを変えたときの例です．

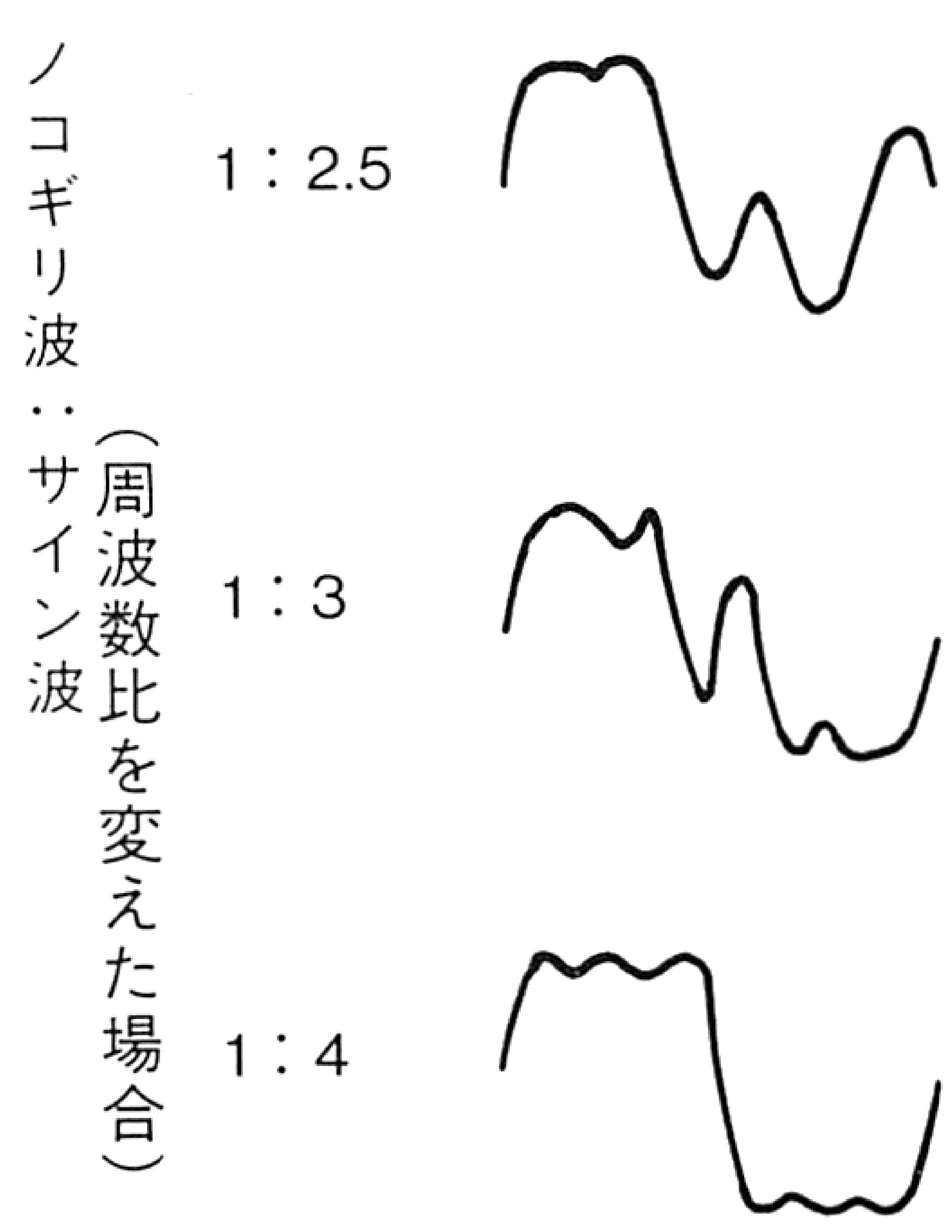


図4.22

まとめとしてここで，特に重要なことを要約しておきましょう．

- ノコギリ波の周波数→読み出される波形の周波数を決定する．
- サイン波の振幅→読み出される波形のかたちを決定する．
- ノコギリ波とサイン波の周波数比→読み出される波形のかたちを決定する．

これでFM音源の基本的な知識が身につきました．

8　オペレータの仕組

FM音源の基本構成は，図4.23のように考えられます．周波数を決めるノコギリ波と，出力の波形を決めるサイン波など(どんな波形でもいい)が加算器で加算され，それで記憶回路を読み出します．記憶回路からの出力は，一たん掛算器を通って出力に出てきます．図4.24のように，振幅が一定の波形を掛算器に加えておいて，エンベロープ[注*]・データ入力に出力される音の輪郭を入れると，波形の振幅はエンベロープ・データに従います．

図4.23の基本構成を描き直してみると図4.25のようになります．そしてこれを“オペレータ”と呼んでいます．

加算器の2つの入力にはそれぞれ名前が付いていて，“ピッチ(＝周波数)・データ入力”と“変調(＝波形)データ入力”になっています．

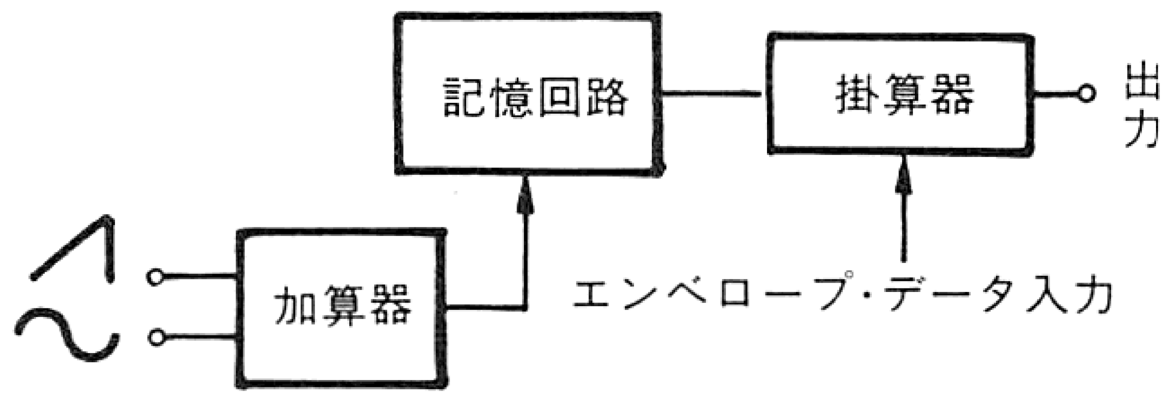


図4.23

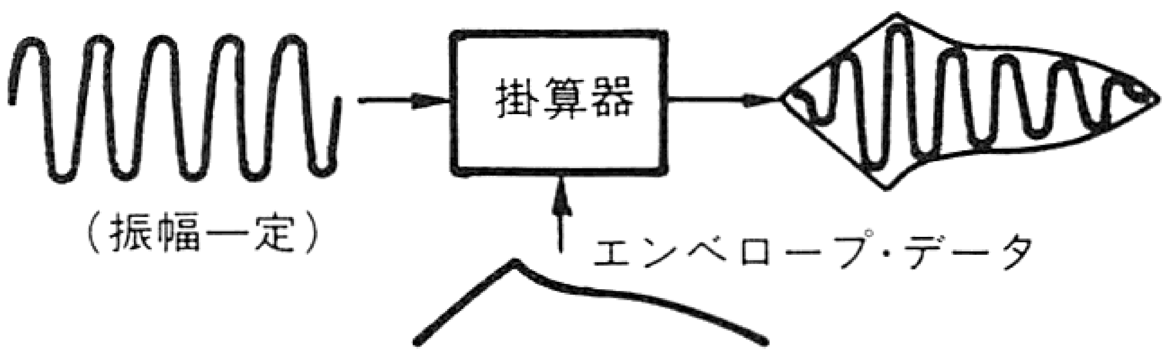


図4.24

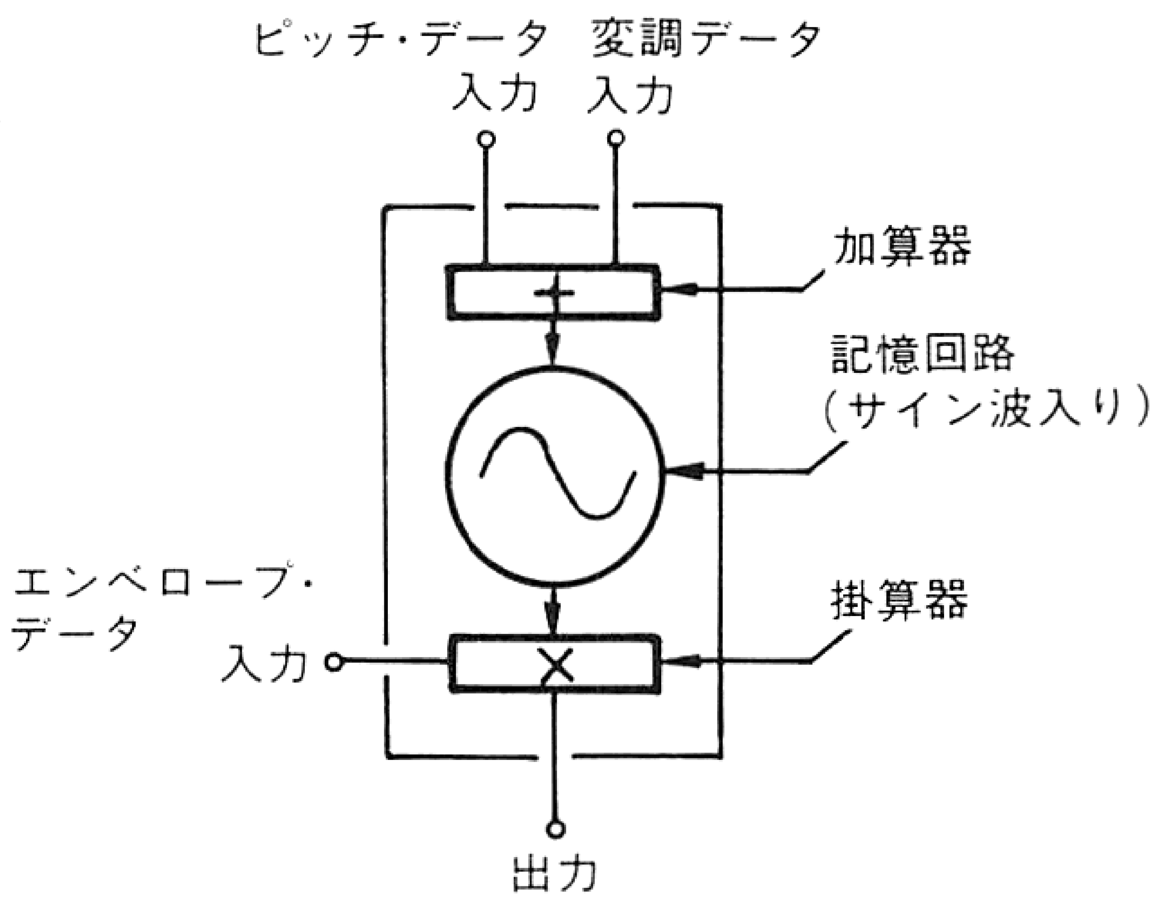


図4.25　オペレータ

注＊：エンベロープ

音の振幅は，発生した瞬間から消えるまで微妙に変化しています．その時間的な変化の様子をエンベロープといいます．エンベロープは，エンベロープデータを与えることによってKey-on(鍵盤楽器にたとえれば，鍵盤を押すことをKey-onといいます．逆に鍵盤から指をはなすことをKey-offといいます.)すると同時にエンベロープジェネレータ(EG)によって自動的に作られます．

9　オペレータの組み合わせ――FM音源での音の作り方

FM音源らしい音の作り方はどんなものなのでしょうか？ 先程出てきた図4.16，4.17の動作をオペレータを使って描き表してみましょう．周波数は，ここでは仮に440Hzとして考えます．

図4.26がその動作でオペレータ１は440Hzのノコギリ波で440Hzのサイン波を作り出しています．エンベロープ・データ入力には何も加えていませんから，出力にエンベロープは付かず，一定振幅のサイン波が出っぱなしになります．

そのサイン波と基になった440Hzのノコギリ波を，それぞれオペレータ２の変調データ入力とピッチ・データ入力に加えるとオペレータ２の記憶回路から読み出された波形は図4.17と同じになります．このように実際のFM音源では，ノコギリ波だけを用意しておけば，どんな波形も作り出すことができます．

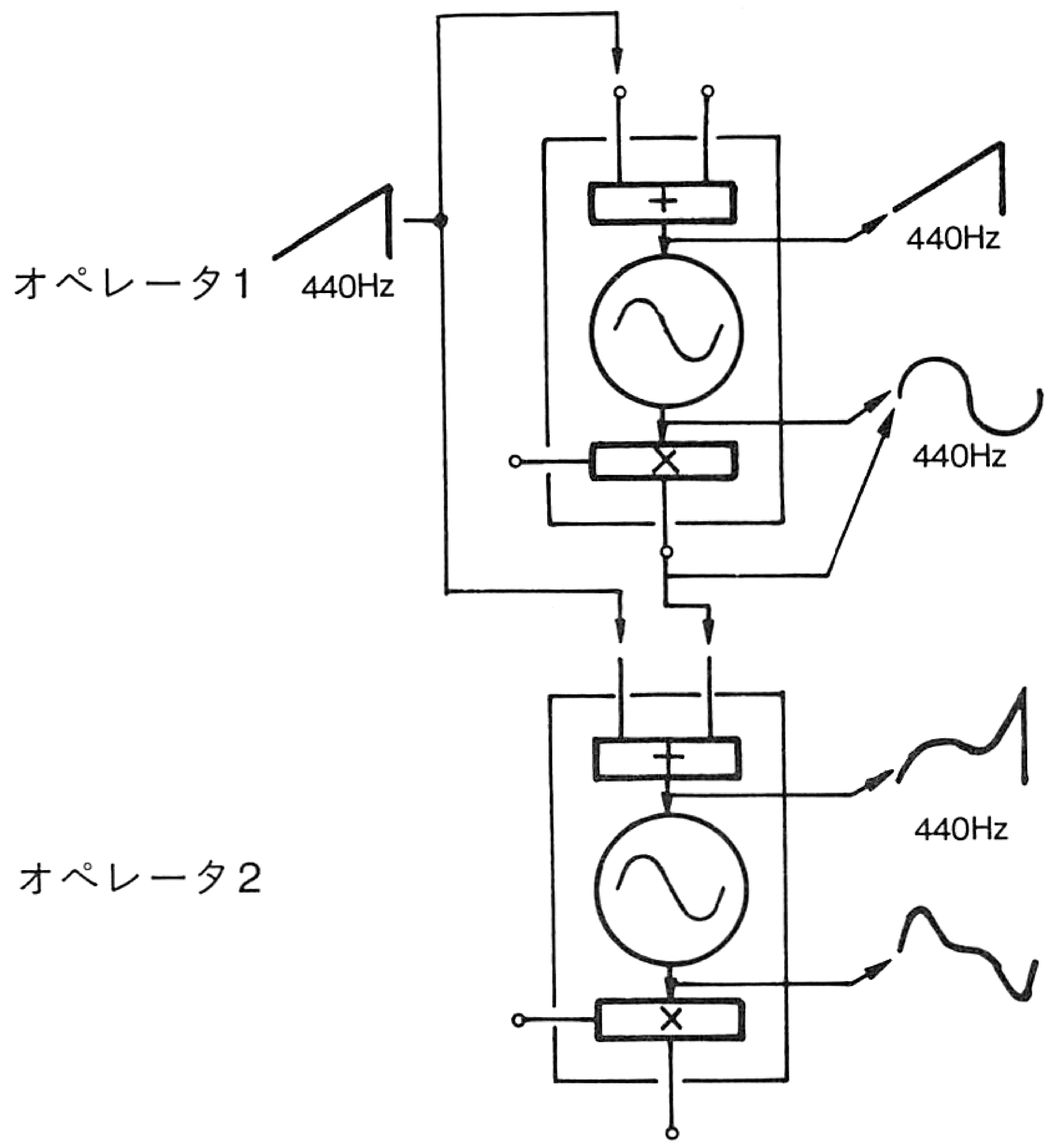


図4.26

また図4.26で，オペレータ１のエンベロープ・データ入力にある波形(の輪郭)を入れると，オペレータ１の出力のサイン波に輪郭が付き(図4.27)，これをオペレータ２に加えると，最終的に得られる波形(オペレータ２の出力)は時間とともに変化し，これは，音色の変化に当たります．つまり，音色に変調をかけたことになります．

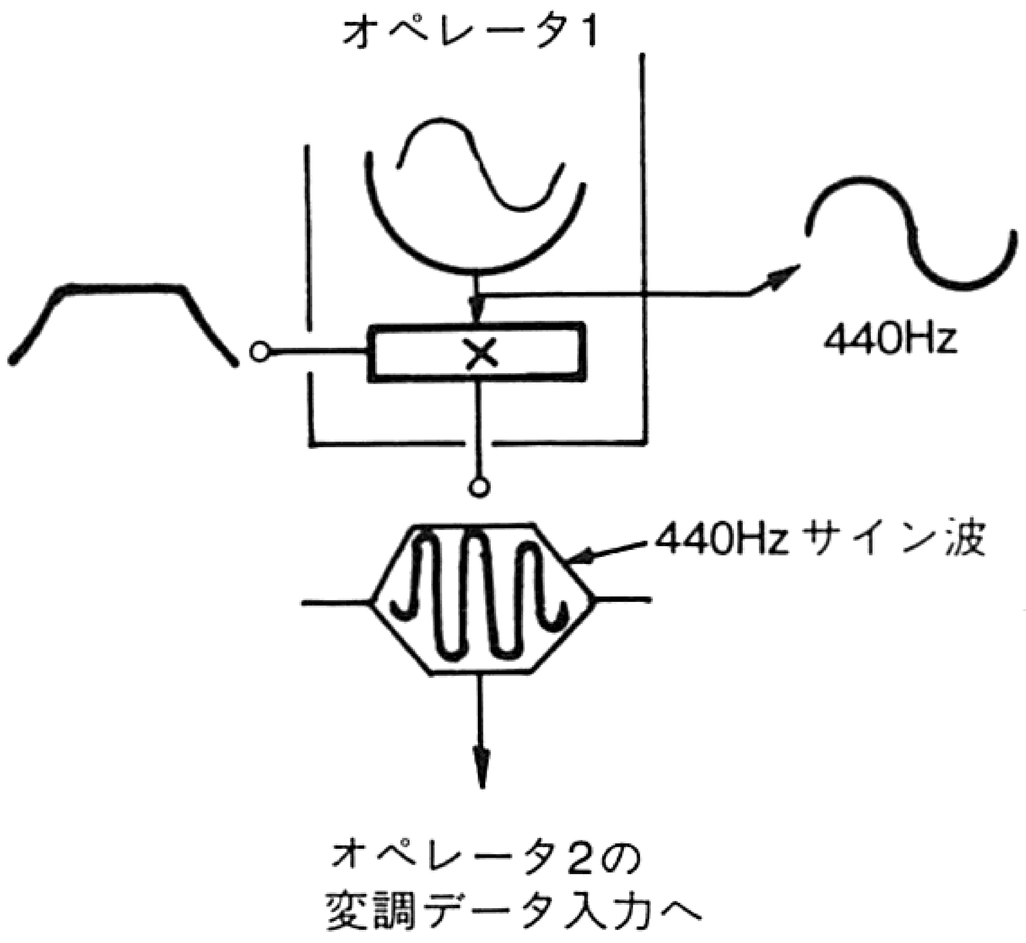


図4.27

これとは別に，エンベロープ・データをオペレータ2に加えると，オペレータ2の出力にも輪郭が付き，これは音量の変化として聴こえます．

FM音源ではこのようにオペレータを組み合わせることによっていろんな音色を作り出しています．

またFM音源ではオペレータを，その使われ方によって2つの種類に分けて考えるようにしています．図4.26の例ですと，オペレータ2は最終的な“音”になる波形を作り出していますが，オペレータがこのような使われ方をした場合，それを“キャリア”と呼びます．また，オペレータ1は，キャリアに与えるための変調用の波形(この場合はサイン波)を作っています．このようなオペレータを，“モジュレータ”と呼びます．

もう一つ，2個のキャリアで音色を作る例を挙げておきましょう．図4.28がそれで，FM音源ではいろんな周波数のノコギリ波を同時に発生させることができることを利用しています．先程の図4.26の例を“直列形”と呼ぶなら，こちらは“並列型”になります．

このようなオペレータの接続の仕方をアルゴリズムといいます．そして，このアルゴリズムが音色を決定する大きな要素となっています．

PC-8001MKⅡSRに内蔵されているシンセサイザーICは，1音に対して4つのオペレータを持ち，図4.29のように8つのアルゴリズムを選ぶことができます．

図4.29では，図4.25のオペレータを箱の形で表し，変調データ入力と出力以外は省略しています．複数の入出力があるものはそれぞれの入出力が加算されていることを意味します．たとえば図4.29の3：のアルゴリズムの場合，オペレータ2とオペレータ3の加算されたものがオペレータ4に入力されます．5：のアルゴリズムの場合，オペレータ2，オペレータ3，オペレータ4の加算されたものが出力されます．

周波数はピッチデータとしてオペレータに与えられます．本体に内蔵されているシンセサイザーICでは，各々のオペレータに独立したピッチデータを与えていません．基本となる周波数があり，その周波数を何倍かしたものが各オペレータに与えられます．そしてその倍数を

Multiple（マルチプル）と呼びます．

たとえば，440HzのAの音をシンセサイザーICに与えたときに，あるオペレータのMultipleが2だった場合，そのオペレータのピッチデータは440Hz×2＝880Hzになります．Multipleの範囲は0～15までの整数で，0を指定した場合周波数は$\frac{1}{2}$になります．

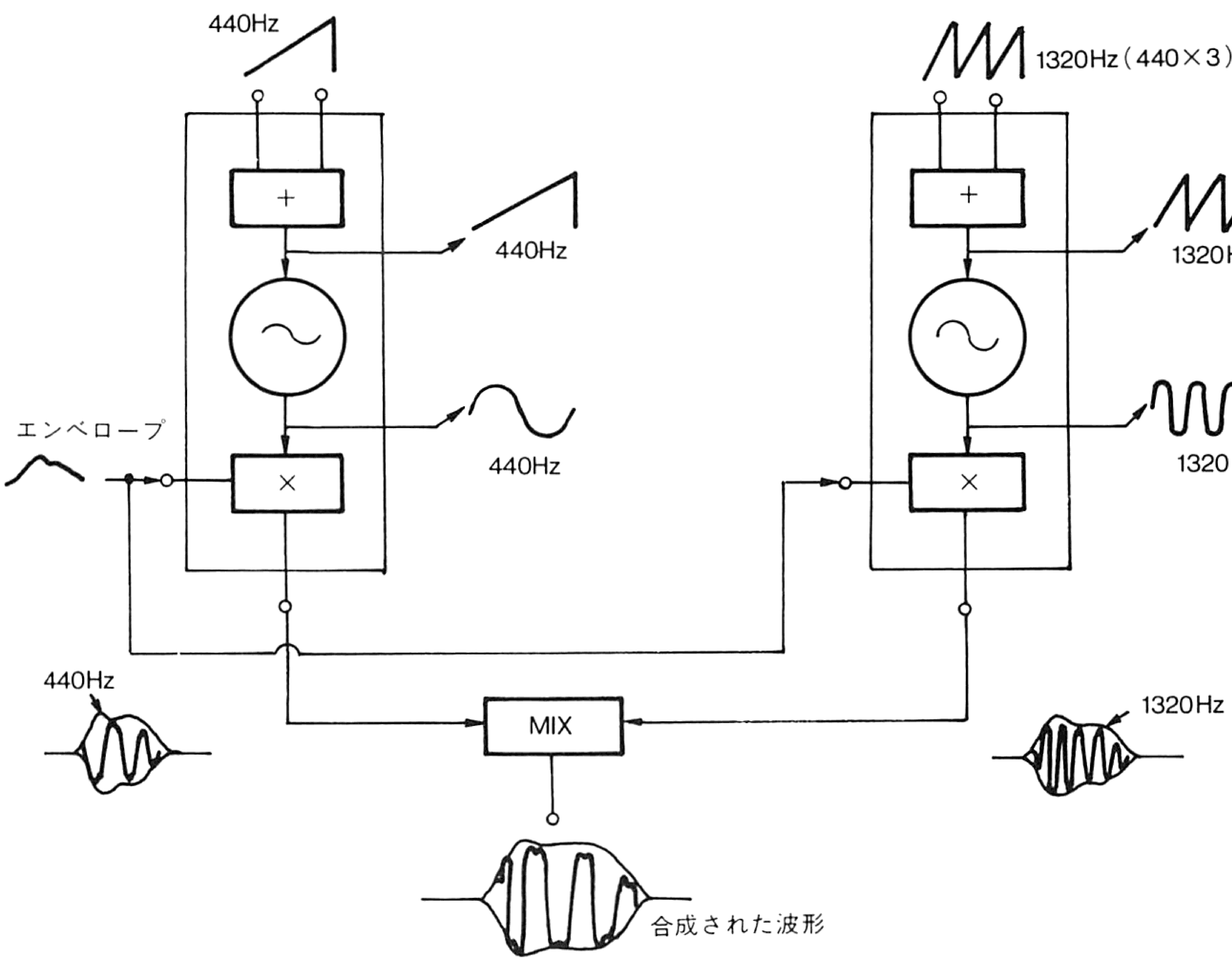


図4.28

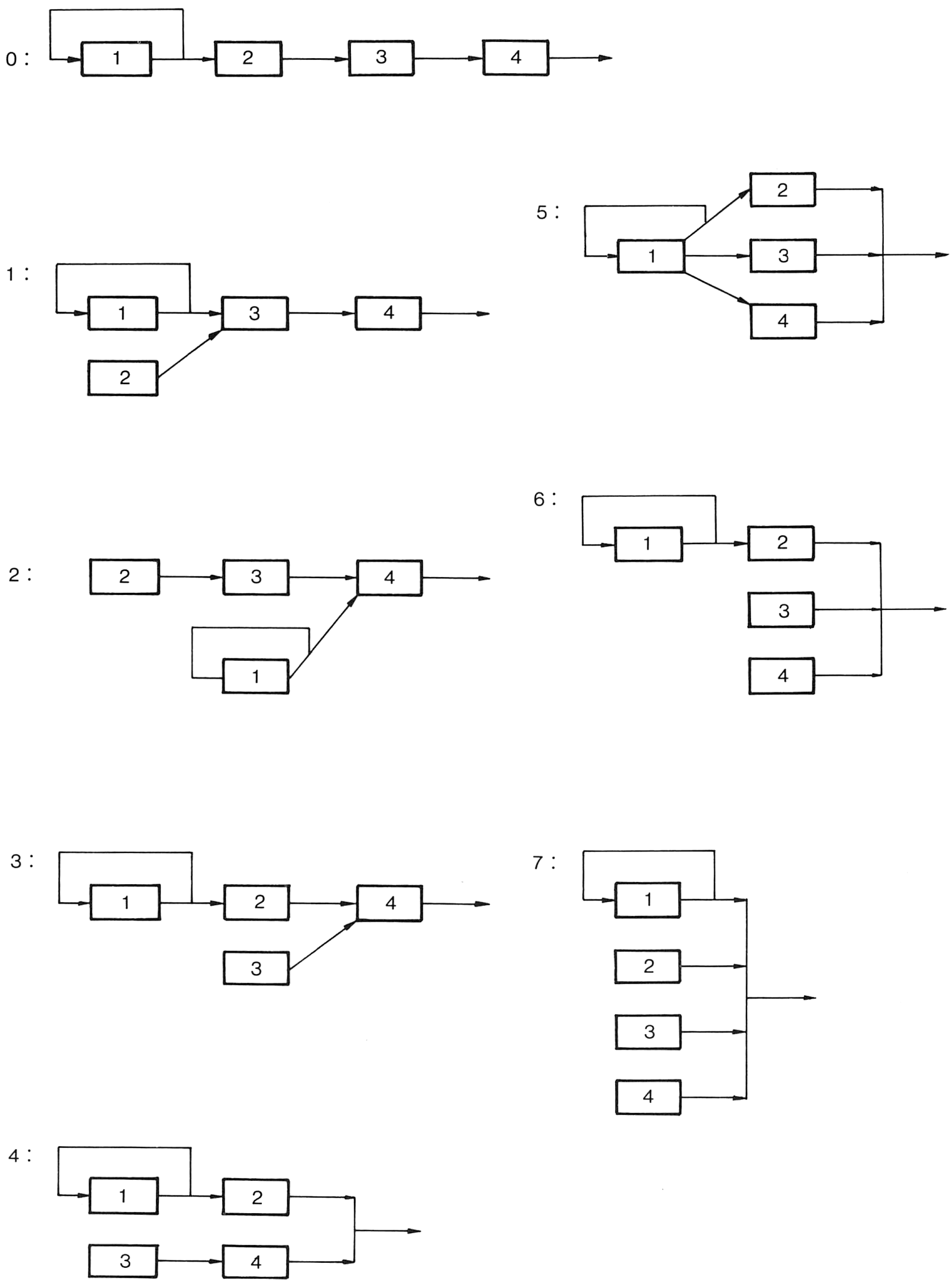


図4.29

資料4.2　実際に音を作る

今までの知識をもとに，実際に音を作ってみましょう．

音作りで最も重要なのは音の時間的変化，つまりエンベロープです．そして，作りたい楽器の特徴をよく理解することも大切なことです．

たとえばピアノであれば，鍵盤を押したとき鋭い音の立ち上がりがあり，その後続けて押していれば除々に音が消えていくエンベロープを持っています．また，倍音の構成も立ち上がりのときに多く，時間が経つにつれて倍音の数は少なくなり，一定の倍音構成に近づいていきます．以上のような特徴をつかんで音作りを進めていきます．

エンベロープを人工的に作る機能をEG(Envelop Generator)といいます．そのパラメータとしてAR(Attack Rate)，DR(Decay Rate)，SR(Sustain Rate)，RR(Release Rate)，SL(Sustain Level)などがあります．

これらを設定していくためには配列変数を使います．この配列変数の各値は，それぞれ特別な意味を持っており，それらの値をVOICE文で音色として設定します．配列変数の意味は2章　VOICE文を参照してください．

まず配列変数を使うためにDIM文で宣言します．

```
dim a%(4,9) ⏎
```

FM音源で音を出すためには，オペレータというものが重要な役割りを果たしています．このオペレータは4つあり，それぞれをいろいろと組み合わせることによって出る音が全く違ってきます．その組み合わせをアルゴリズム(Algorithm)といい8つのパターンがあります．(2章　VOICEのアルゴリズム表を参照してください.)

4つあるオペレータはモジュレータとキャリアに分かれ，これはアルゴリズムによってその対応が変わります．「9　オペレータの組み合わせ――FM音源での音の作り方」で述べたように，キャリアは出力する周波数がそのまま音の周波数になり，モジュレータは出力する周波数が変調信号としてキャリアに対して出力されます．

キャリアとモジュレータのパラメータの効果は，次のようになっています．

項　目	関与するパラメータ	設定値による音の変化 値 MIN ↔ MAX
キャリアの出力レベル	各オペレータのOutput Level (AR, DR, SR, RR, SL)	音量大 ↔ 音量小 (注: Levelは0が最大となります)
モジュレータの出力レベル		丸い音色↔明るい音色
モジュレータのフィードバックレベル	Feedback	普通の音色↔鋭い音色ノイズ
キャリアの周波数	各オペレータのDetune, Multiple	ピッチ低↔ピッチ高
モジュレータの周波数		近い倍音↔離れた倍音

次に，8つのアルゴリズムの特性を説明します．

Ⅰ）キャリア数1のアルゴリズム(番号0～3)

1つのキャリアに対して残りのすべてのオペレータがモジュレータとなるわけですから，強力に明るい音になります．またこのアルゴリズムは極端な音色変化だけでなく，モジュレータの出力レベルを控え気味にすれば，複雑な波形を持つ微妙な音色を作ることも可能であり，ソロ楽器向きといえます．

Ⅱ）キャリア数2のアルゴリズム(番号4)

キャリアが2つのアルゴリズムは，音作りの結果を想像しやすいわりに，幅広い音色が作れるオールマイティなパターンです．2つのキャリアのピッチをずらしてコーラス効果を出したり，一方で基本的な音色を作り，他方で味付け的な効果(たとえば，息の音)を出したりすることも可能です．

Ⅲ）キャリア数3，4のアルゴリズム(番号5～7)

厚みのある落ち着いた音色を作るのに向いています．すべてのキャリアのピッチを少しずつずらしてやると，多数の楽器を同時に弾いているようなコーラス効果が作れます．またアルゴリズム7のように4つのオペレータの出力をすべて加えあわせるパターンは，オルガン系の音色を作るのに最適です．

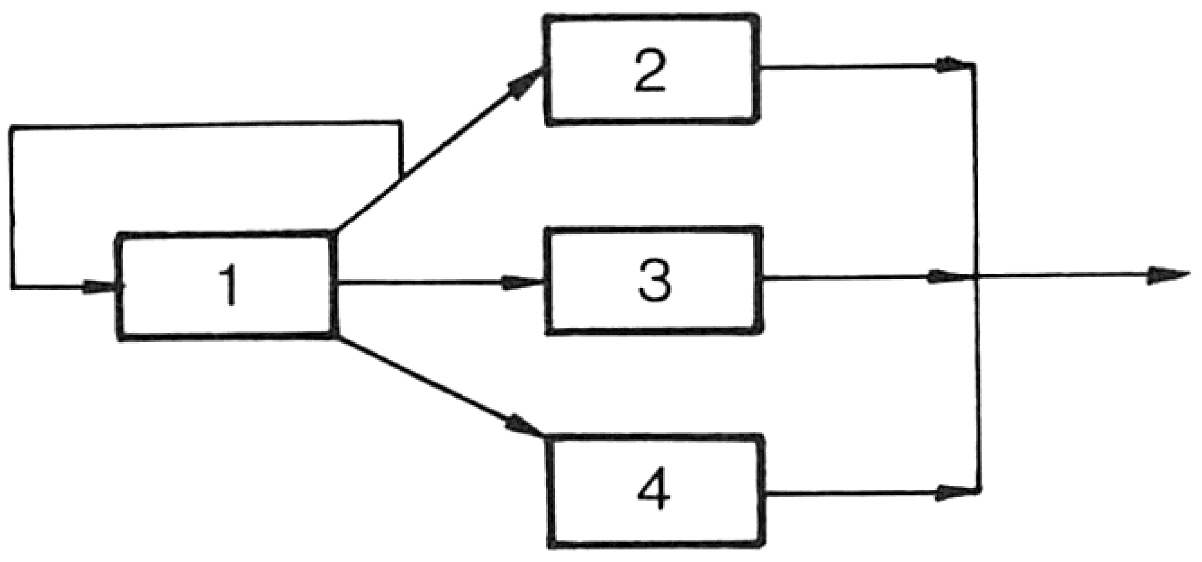
それでは，これからエレクトリックオルガンの音を作ることにしましょう．

そこでまず初めに，これら4つのオペレータをONにして使用可能(音を出せる状態)にするために，次のようにしてください．

a%(0,1)=15 ↵

この値の意味は，2章　VOICE文の配列変数の意味を参照してください．

次にアルゴリズムを選ぶわけですが，前にも述べたように，オルガン系はアルゴリズム7が良いのですが，ノイズを付けてボリューム感のある音を作るためにアルゴリズム5を選びます．



アルゴリズム5

モジュレータであるオペレータ1がすべてのキャリア(オペレータ2，3，4)に対して変調をかけることによってノイズを付けることができます．

アルゴリズムの設定は配列(0, 0)で行い，フィードバック(Feedback……自分自身に対して変調をかける度合を設定する)の値も一緒に設定します．このフィードバックはオペレータ1に働くもので，先ほども述べたように歪んだ感じを出してボリュームある音にするために，ここでは最大値の7にしておきます．この設定方法は，2章 VOICE文のFeedback/Algorithmを参照してください．

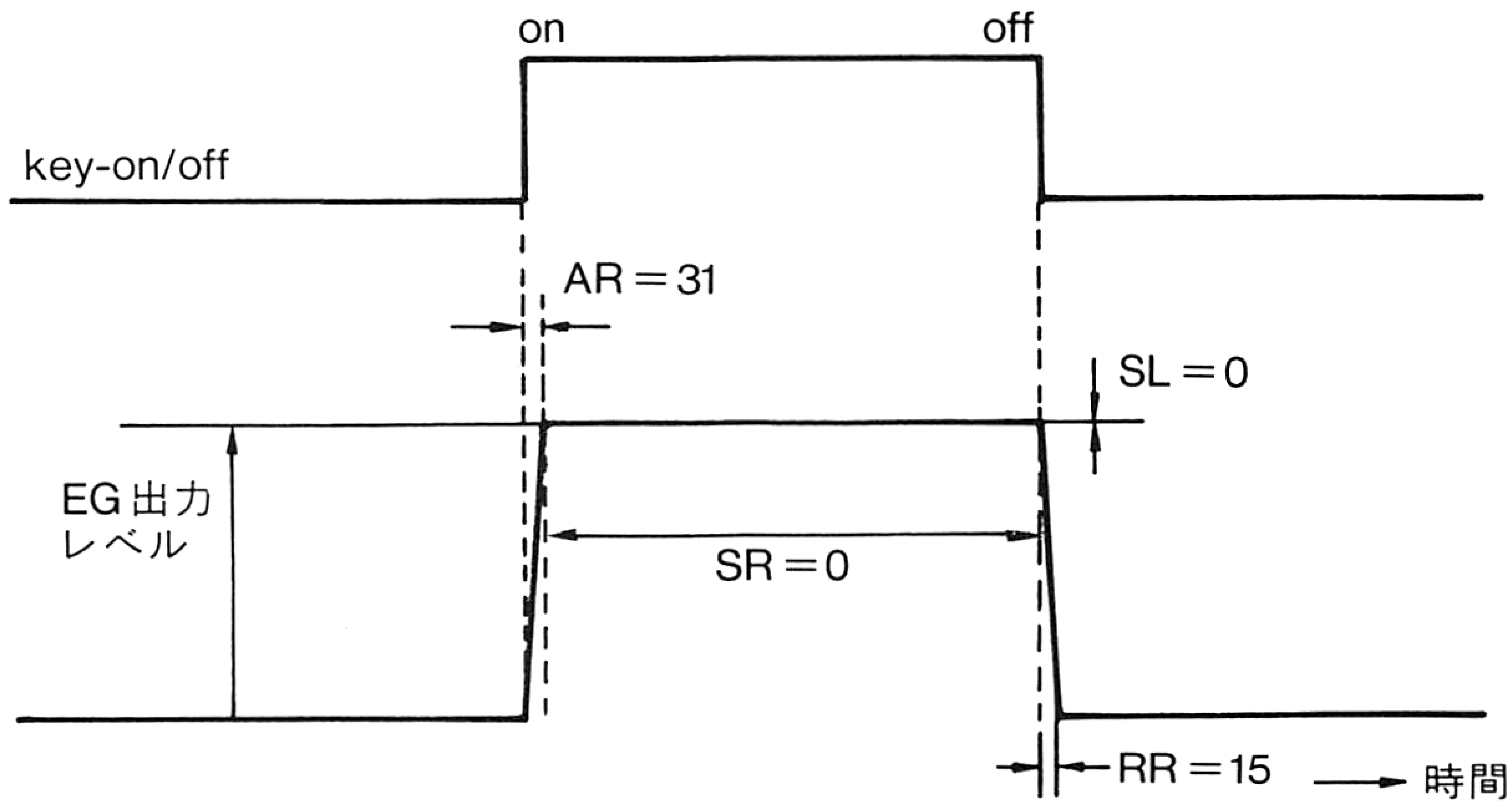
```
a%(0,0)=7*8+5 ⏎
```

これでフィードバックとアルゴリズムの設定ができました．

次に音を出すためにはAR(Attack Rate)を設定します．ARを設定しない場合は，音の立ち上がる速さが設定されないので音が出ません．またRR(Release Rate)も設定します．このRRはKey-offしてからEG出力レベルが減衰していく速さを決めます．このRRを設定しなければ(RR=0のとき)，かなり長い間音が続きます．もしこのような状態になったときは，違う音色(RRが設定されている音色)を出して前の音を消してください．とりあえず両方とも最大値にしておきます．

```
for op=1 to 4:a%(op,0)=31:next op ⏎
for op=1 to 4:a%(op,3)=15:next op ⏎
```

ここでオペレータ1～4の現在のエンベロープを概念的に表すと下図のようになります



音を出すために，VOICE文でデータを音色として設定しましょう．

```
voice a% ⏎
play "v15o4cdefg" ⏎
```

どうですか．かなり激しい雑音がしたと思います．これはモジュレータのOutput Levelが高すぎるために起こります．モジュレータのOutput Levelが高すぎると，そのモジュレータがキャリアに対してかける変調度が大きくなりすぎて雑音のような音になります．

一方，キャリアの方のOutput Levelは他のキャリアとの兼ね合いもありますが，キャリアが1つだけの場合は，出力される音量の大小に関係します．

モジュレータのOutput Levelは作ろうとする音質にもよりますが，最大でも10くらいにとどめておいたほうがよいでしょう．

ここでは，モジュレータはオペレータ1だけで，オペレータ1にはフィードバックを最大にかけているので，Output Levelを30に設定しておきます．

```
a%(1,5)=30 ⏎
```

もう一度音を出してみましょう．次のように入力してください．

```
voice a%
play "v15cdefg" ⏎
```

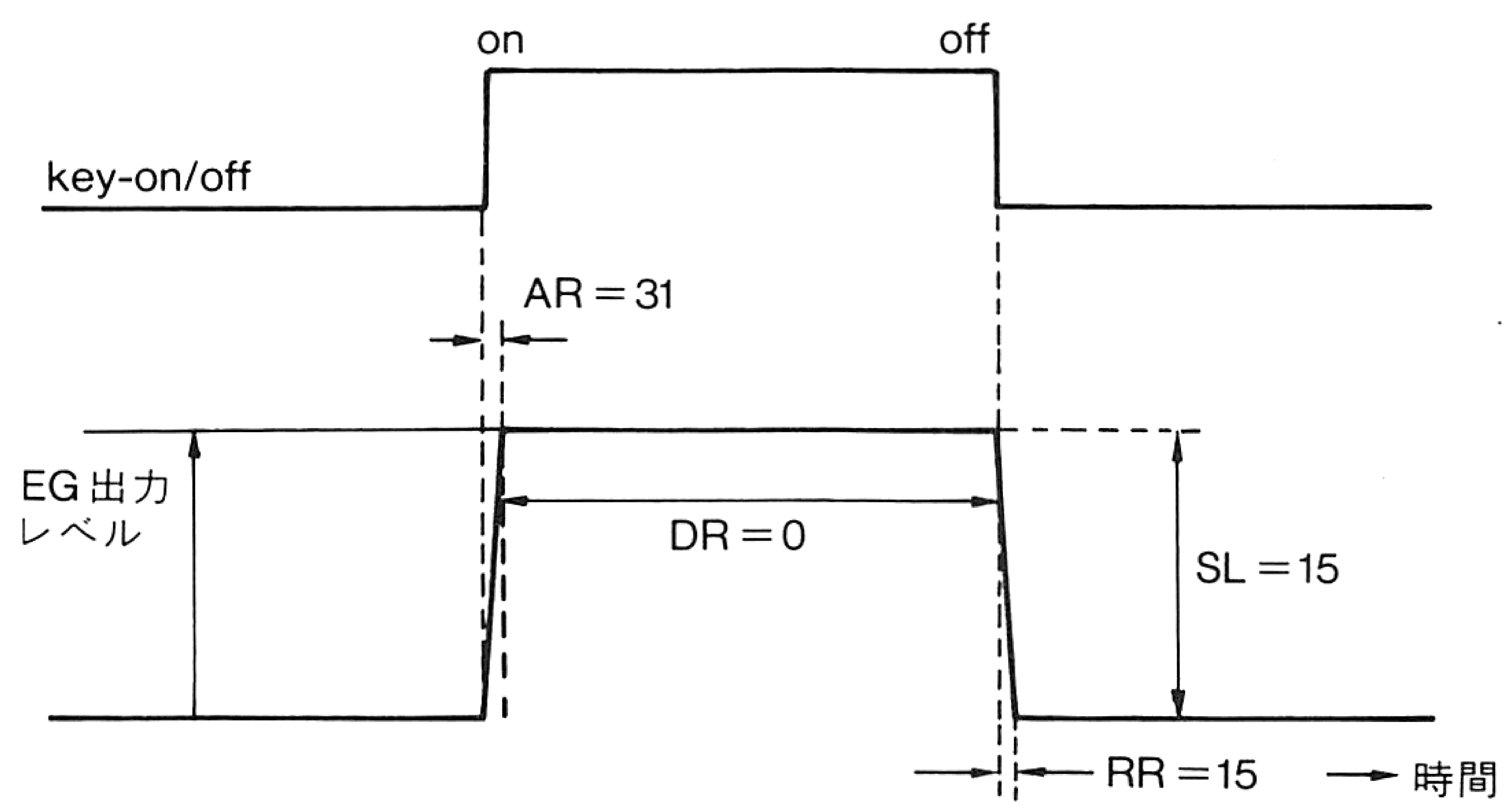
いくらかいい音になったと思います．まだ実際のエレクトリックオルガンの音とは全く違っていますが細かい修正はあとにして，先に次のパラメータを設定することにします．

次はDR(Decay Rate)を設定します．Key-onされてARの値によりEG出力レベルが最高になってからこのレベルが下がっていく速さを，DRとSR(Sustain Rate)で決めます．このDRとSRはSL(Sustain Level)によって分かれます．EG出力レベルが最高からSLに達するまでの速さがDRで，SLから出力が0あるいはKey-offされるまでがSRの値になります．このSLを0(減衰量を0とします．したがって出力レベルは最高値)にするとEG出力レベルが最高に達した時点からSRになりますので，DRの値は意味を持たなくなり，またそれとは逆にSLを15にすると，EG出力レベルが0になるまでDRの値により減衰していくのでSRの値は意味を持たなくなります．

各オペレータのSLを15にします．

```
for op=1 to 4:a%(op,4)=15:next op ⏎
```

オペレータ1～4の現在のエンベロープを概念的に表すと下図のようになります．



これでだいたいの準備ができました．それではこれからオペレータを分けて部分的に修正して音を作り上げていきましょう．

まず，オペレータ４だけONにします．（このオペレータと設定値の関係は２章　VOICE文を参照してください）

```
a%(0,1)=8 ⏎
```

現在，Multiple(周波数比)の値が０に設定されています．これを１にしておきます．このMultipleは，入力された周波数を何倍にするかを決めるものです．

```
a%(4,7)=1 ⏎
```

これで音を出してみましょう．

```
voice a% ⏎
play "cdefg" ⏎
```

あまり音に表情が感じられませんね．これは正弦波によって音が作られているからです．

そこで，次のキャリアであるオペレータ３をONにしてハモった感じを出すためにMultipleを２にして音を出してみましょう．

```
a%(0,1)=4:a%(3,7)=2 ⏎
voice a% ⏎
play "cdefg" ⏎
```

音程は高くなりましたが，やはりこれも正弦波１つなので単純な音ですね．今度はオペレータ３と４をONにして音を出します．

```
a%(0,1)=12 ⏎
voice a% ⏎
play "cdefg" ⏎
```

今度は，原音のオペレータ４と倍音のオペレータ３の音がミックスされてオルガンらしい音色に近づいてきました．次に最後のキャリアであるオペレータ２で高い周波数を出すために，Multipleを10に設定します．

このとき，オペレータ２をONにするのを忘れてはいけません．

```
a%(0,1)=14:a%(2,7)=10 ⏎
voice a% ⏎
play "cdefg" ⏎
```

高音部は出るようになりましたが，音程がくるっているように聞こえるのでもう少し下げま

す．

```
a%(2,7)=8 ⏎
voice a% ⏎
play "cdefg" ⏎
```

これで良くなりましたね．次にモジュレータであるオペレータ１をONにして音を出してみます．

```
a%(0,1)=15 ⏎
voice a% ⏎
play "cdefg" ⏎
```

多少いびつな音に聞こえますのでオペレータ１のMultipleを５にしてOutput Levelを少し下げます．

```
a%(1,7)=5:a%(1,5)=40 ⏎
voice a% ⏎
play "cdefg" ⏎
```

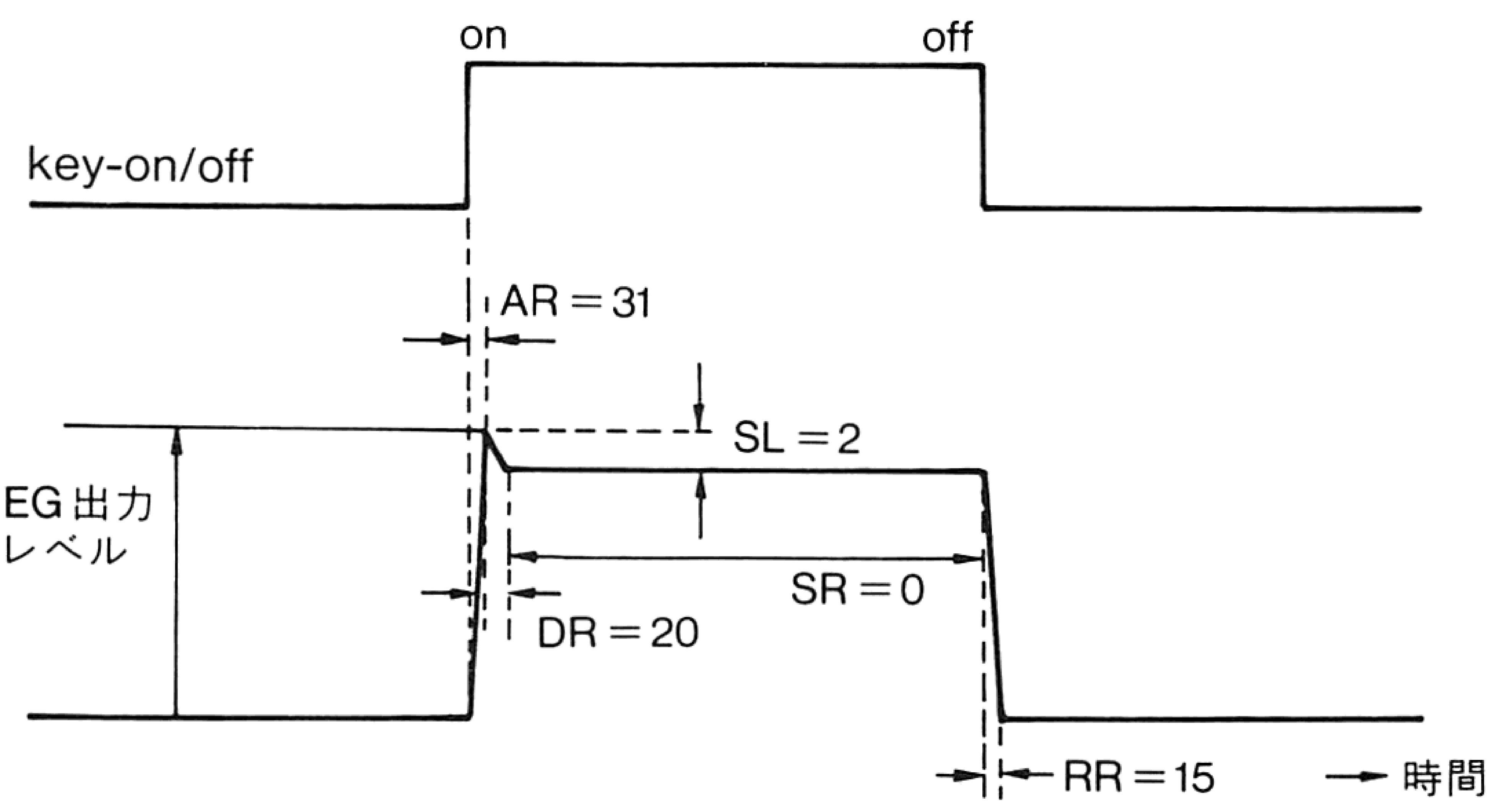
ずいぶんとオルガンらしい音になってきました．しかし，まだ歪んだ感じがしますのでオペレータ２，３のOutput Levelを下げます．

```
a%(2,5)=20:a%(3,5)=30 ⏎
voice a% ⏎
play "cdefg" ⏎
```

まだ歪んだように聞こえますね．これはオペレータ１による変調がオペレータ2，3，4全部に同様にかかっているためです．そこでオペレータ１の出力レベルを音が立ち上がった状態から少し下げるようにします．それには，SLを２に設定すればよいでしょう．そしてDRが０ではEG出力レベルが下がらないので，立ち下がりを少し速めの20に設定します．

```
a%(1,4)=2:a%(1,1)=20 ⏎
voice a% ⏎
play "cedef" ⏎
```

オペレータ1の現在のエンベロープを概念的に表すと下図のようになります.



これで少し丸みが出てきて，エレクトリックオルガンの感じになってきました．あとはもう少し音の幅を持たせるために，デチューン(Detune)[注*]を設定して，周波数のピッチをずらします.

a%(1,8)=3:a%(2,8)=−1 ⏎

a%(3,8)=−3:a%(4,8)=3 ⏎

そのあとKeyboard Rate Scaling Depth[注**]を設定して，音程が高くなるほど各Rateによる変化速度が速くなるようにします．オペレータ2，3，4の各Rate値が最大か，あるいは0で設定されていなければKeyboard Rate Scaling Depthを設定しても意味がないので，ここではオペレータ1だけ3にします.

a%(1,6)=3 ⏎

そして最後に，LOF(Low Frequency Oscillator)のパラメータを設定してビブラート(音程を細かく変化させる)やトレモロ(音量を細かく変化させる)などの効果を付けます．しかし，このLFOをあまり深くかけすぎると音程がくるうことがあるので注意してください.

このエレクトリックオルガンの音には，ビブラートだけでトレモロは付けませんが，トレモロをかけるためのパラメータの設定は，LFOスピードが1500〜2500くらい，Ams(Amplitude Modulation Sensitivity)を3〜8，Amd(Amplitude Modulation Depth)を低くするとなめらかな効果が得られます．トレモロをかけるときは，最初にこの程度の値を設定しておいて，あとは音色によって修正していくとよいでしょう.

ここではビブラートをかけるために，ピッチ(Pmd(Pitch Modulation Depth)， Pms(Pitch Modulation Sensitivity))を変えます．細かいビブラートをかける場合，LFOスピードは3000〜4000くらいがいいでしょう.

エレクトリックオルガンは，それほど強くビブラートをかける必要はないので，Pmsを５，Pmdは10に設定します．

```
a%(0,4)=3000 ⏎
a%(0,7)=5 ⏎
a%(0,5)=10 ⏎
```

次にWave formですが，ビブラートには三角波が適していますので，２を設定して音を聞いてみましょう．LFOはやや高めで長い音にするとわかりやすくなります．

```
a%(0,2)=2 ⏎
voice a% ⏎
play "l1o6c" ⏎
```

もう少しはっきり表すためにPmsを上げます．しかしこのままでは，かかりすぎるのでPmdを少し下げることにします．次のようにしてください．

```
a%(0,7)=6:a%(0,5)=8 ⏎
voice a% ⏎
play "l1o6c" ⏎
```

これでほど良い効果が得られたと思いますが，もっと長い音を出してLFOの効果を確認しましょう．それには，"@W"を使うと便利です．

この"@W"は，指定されたときの状態を指定された長さだけ維持します．わかりやすく説明しますと，"@W"が指定されたときにKey-on（音が出ている）状態であれば，そのままKey-onされていて，音が出ていますが，指定されたときにKey-offされていれば，そのままKey-offの状態，つまり音が出ない状態が続きます．よって，音程（C～B）の後に指定しても，この音程（C～B）は，Key-offしてから次のMMLパラメータを実行しますので，結局は"R"（レスト）と同じ効果しか生みません．そこで長い音をずっと出す場合には，"Y"を使ってOPNレジスタに直接データを設定してKey-on状態を作り，その後に"@W"を指定します．OPNレジスタのKey-on, offを設定するレジスタ番号は40（28H）ですので，そこへ240（F0H）を設定するとKey-on状態になります（OPNレジスタについては資料4.3　シンセサイザーICの構造を参照してください）．

```
play "y40,240@w1@w1@w1@w1" ⏎
```

以上でエレクトリックオルガン音の出来上がりです．

今回挙げた例は，全く何もない状態から音色を作っていきましたが，作ろうとする音に近い音，あるいは特徴の似ているものを既存の音色から見つけだして，その音をもとにして修正を

重ね音を作るようにすれば，もう少し簡単に音を作ることができます．

その方法は，似ている音色をVOICE COPY文で配列変数にパラメータをコピーして，その後は各パラメータの値と音とを比較しながら，今まで説明してきたように修正してください．

注＊：デチューン(Detune)

FM音源のオペレータのピッチを微妙にずらすことによって音色にうねりや深みを与えることができます．

オルガン，エレクトリックピアノ，ストリングスに，この機能を与えることによって，より自然な楽器の感じが得られます．

注＊＊：　キーボードレイトスケーリングデプス(Keyboard Rate Scaling Depth)

音程によってエンベロープの各Rateを変化させます．自然の楽器のエンベロープはピアノの低音部と高音部を聞いてもわかるように音程によって微妙に異なっています．

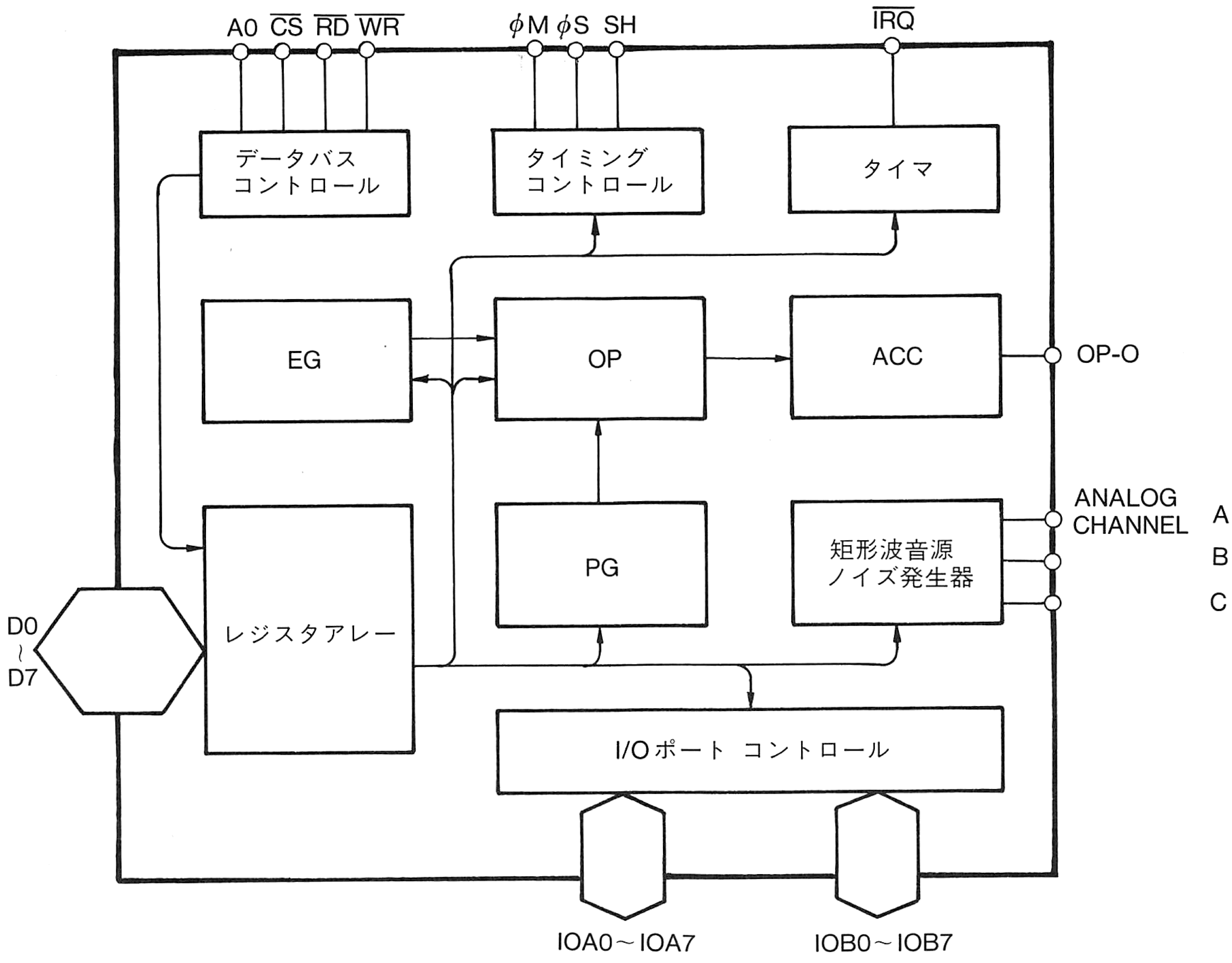
このキーボードレイトスケーリングデプスを設定することによって高音部での各Rateが速くなり，より自然な楽器の音に近い感じが得られます．

資料4.3　シンセサイザーICの構造

OPNはFM音源，SSG音源，2つのタイマAとBから構成されます.ここでは，FM音源，SSG音源を中心に説明します.

資料4.3.1　シンセサイザーICの概要

FM音源部は，FM音源をコントロールするデータのレジスタ(REG)，フェイズ・ジェネレータ(PG)，エンベロープ・ジェネレータ(EG)，FMの演算を行うオペレータ(OP)及び各スロットの和を求めるアキュームレータ(ACC)から成り立っています.



ブロックダイアグラム

(a)　**レジスタ(REG)**

レジスタには，FM音源に対するレジスタと，SSG音源に対するレジスタの2種があります.

FM音源部のレジスタのエリアは，表1のアドレスマップで示されるように20HからB2Hまでの147バイトのなかで割り当てられています．ここでいうアドレスとは，OPN内で各レジスタに割り当てられたサブアドレスであり，このアドレスの割り当てはデータビットD0～D7で行います．したがって，あるレジスタにデータを書き込む場合には，データに先立ってサブアドレスのデータを送り，次にFMデータを送ることになります．サブアドレスについては，一度あるアドレスを書き込むと，次にアドレスを書き換えるまで，そのアドレスは有効であり，同一のアドレスを何度もアクセスする場合は，アドレスの書き込みは最初に設定するだけで，その後については不要です．アドレスとFMデータの切り替えは，A0入力で行われ，A0＝"0"すなわちLowレベルのときは，データビットD0～D7上のデータはサブアドレスを意味し，A0＝"1"すなわちHighレベルのときはFMデータとなります．この手法は，FM音源部のデータが書き込みだけしかできないのに対して，SSG部ではレジスタの内容を読むこともできるという点を除けば，SSG部においても同様にレジスタのアクセスを行うことができます．

なお，全レジスタとも初期設定（イニシャル・クリア：$\overline{\text{IC}}$端子＝"0"）のときに，オール0にされます．

表4.1　OPNアドレスマップ１

※WRITE DATA（FM音源部）

ADDRESS	DATA				COMMENT
21H	TEST				LSIのTEST-DATA
24H	TIMER-A				TIMER-Aの上位8bits
25H		TIMER-A			TIMER-Aの下位2bits
26H	TIMER-B				TIMER-BのDATA
27H	MODE	RESET B A	ENABLE B A	LOAD B A	TIMER-A/BのControl及び3CHのMode
28H	SLOT		CH		Key-on/off
2DH					プリスケーラをSet
2EH					1/3, 1/6分周の選択
2FH					分周器を1/2分周にセット
30H–3EH		DT	MULTI		Detune / Multiple（33H, 37H, 3BHのAddressは無し）
40H–4EH		TL			Total Level（43H, 47H, 4BHのAddressは無し）
50H–5EH	KS		AR		Key Scale / Attack Rate（53H, 57H, 5BHのAddressは無し）
60H–6EH		DR			Decay Rate（63H, 67H, 6BHのAddressは無し）
70H–7EH		SR			Sustain Rate（73H, 77H, 7BHのAddressは無し）
80H–8EH	SL	RR			Sustain Level / Release Rate（83H, 87H, 8BHのAddressは無し）
90H–9EH		SSG-EG			SSG-Type Envelop Control（93H, 97H, 9BHのAddressは無し）
A0H A1H A2H	F-Num. 1				F-Numbers / Block
A4H A5H A6H		Block	F-Num. 2		F-Numbers / Block
A8H A9H AAH	3CH＊F-Num. 1				3CH-3slot F-Numbers / Block
ACH ADH AEH		3CH＊Block	3CH＊F-Num. 2		3CH-3slot F-Numbers / Block
B0H B1H B2H		FB	CONNECT		Self-Feedback / Connection

表4.2　OPNアドレスマップ2

※READ/WRITE DATA(SSG音源部)

ADDRESS					COMMENT
00H	Fine Tune				Channel-A Tone Period
01H		Coarse Tune			
02H	Fine Tune				Channel-B Tone Period
03H		Coarse Tune			
04H	Fine Tune				Channel-C Tone Period
05H		Coarse Tune			
06H		Period Control			Noise Period
07H	IN/OUT IOB IOA	/Noise	/Tone		/ENABLE
08H		M	Level		Channel-A Amplitude
09H		M	Level		Channel-B Amplitude
0AH		M	Level		Channel-C Amplitude
0BH	Fine Tune				Envelop Period
0CH	Coarse Tune				
0DH		C	ATT	ALT HLD	Envelop Shape, Cycle
0EH	I/O Port-A				I/O Port Date
0FH	I/O Port-B				

※READ DATA

ADDRESS				COMMENT
00H～0FH	BUSY		FLAG B A	Status

資料4.3.2　FM音源部

(a—1) TEST : ADDRESS[21H]

このアドレスは，OPNをテストするために設けられたのであり，常に"0" レベルにしておかなければなりません．

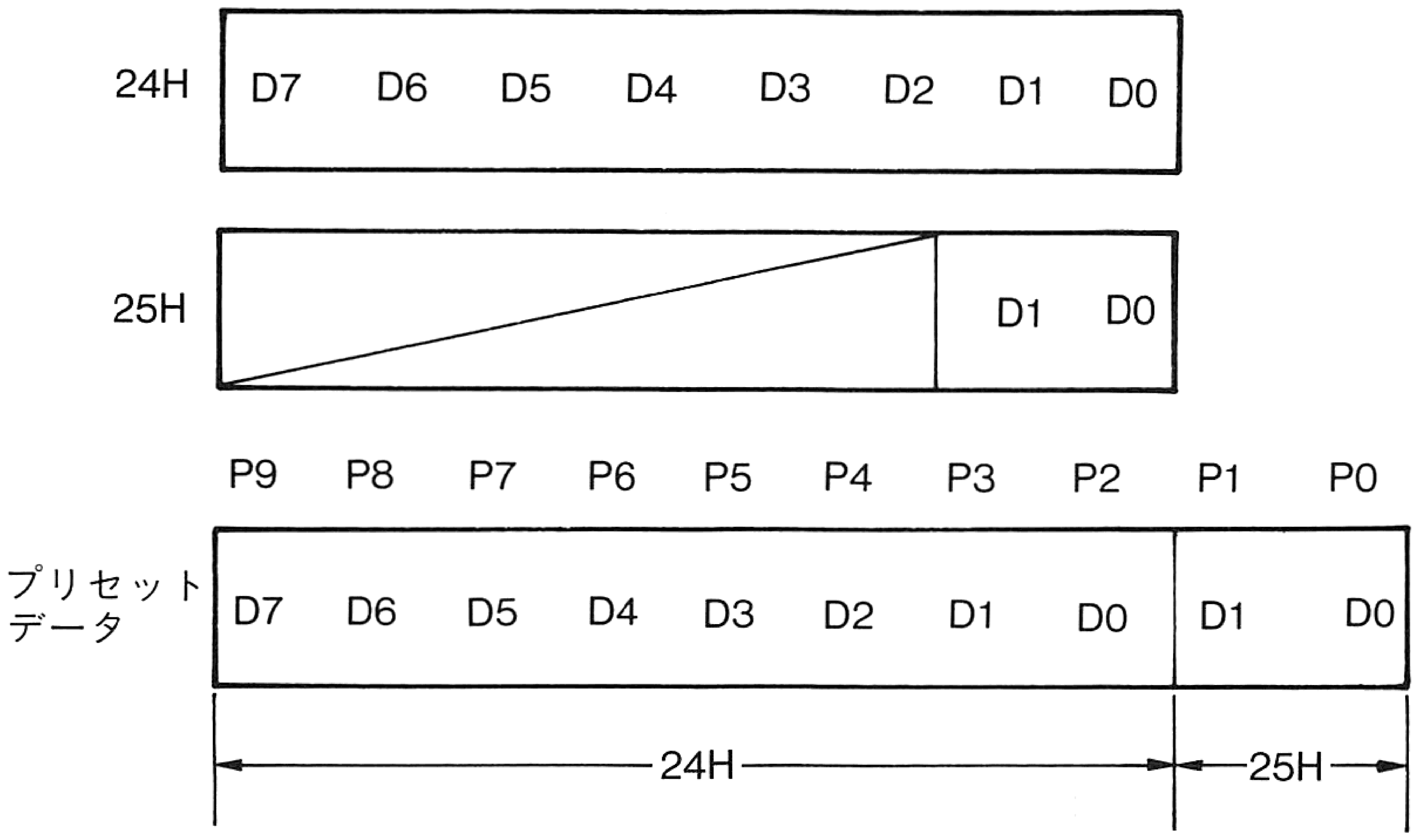
(a—2) TIMER

タイマは2個持っており，１つは10ビット・プリセッタブル・タイマ(TIMER—A)で，もう一つは８ビット・プリセッタブル・タイマ(TIMER—B)です．各タイマは始動・停止及びフラグの制御が可能です．

(I) TIMER—A : ADDRESS[24H]・[25H]

タイマAは，アドレス24Hと25Hの10ビットからなり，この10ビットの値をプリセット値としてカウンタを動かします．そして，カウンタがオーバーフローを生じたときにタイマAのフラグを立て，同時にプリセット値をロードします．

タイマAは通常のタイマ機能以外にCSMのコントロールとしても働きます．この場合は，オーバーフローが生じたときのみです．チャネル3の各スロットをONにして，チャネル3の全スロットの音を出します．これにより複合正弦波合成による音声合成が可能になります．



T 0VA(ms)=12＊(1024—NA)/f FM(kHz)

NA : $P9*2^9+P8*2^8+P7*2^7+\cdots+P1*2^1+P0$

f FM : プリスケーラの項(a—4)参照

例)　T 0VA(MAX)=20.48ms(@ f FM=600kHz)

T 0VA(MIN)=0.02ms(@ f FM=600kHz)

(Ⅱ) TIMER－B : ADDRESS[26H]

タイマBは8ビットのプリセッタブル・タイマであり，タイマと同様にカウンタのオーバーフロー時にフラグを立てます．

26H	D7	D6	D5	D4	D3	D2	D1	D0
	P7	P6	P5	P4	P3	P2	P1	P0

T 0VB(ms)＝192＊(256－NB)/f FM(kHz)

NB : P7＊2^7＋P6＊2^6＋ … ＋P1＊2^1＋P0

f FM : プリスケーラの項(a－4)参照

例)　T 0VB(MAX)＝81 . 92ms(@ f FM＝600kHz)

T 0VB(MIN)＝0 . 32ms(@ f FM＝600kHz)

(Ⅲ) タイマ・コントロール及びチャネル3のMODE設定：ADDRESS[27H]

アドレス27Hでは，タイマA，タイマBの始動・停止・フラグの制御をD0～D5のビットで行います．また，D6, D7の2ビットでチャネル3のMODEを決定します．

27H	D7	D6	D5	D4	D3	D2	D1	D0

各ビットの意味

D0 : タイマAの始動・停止の制御をします．このビットに"1"が立ったときに，その立ち上がりで，タイマAのプリセット値をロードしてタイマAを動かします．そして，このビットが"0"になるとタイマAは停止します．

D1 : タイマBの始動・停止の制御ビットで，働きはタイマBに対してD0と同様なことを行います．

D2 : このビットに"1"が立つと，タイマAのオーバーフローを受けてAフラグに"1"を立てます．"0"のときはオーバーフローは受け付けません．

D3 : このビットはタイマBに対してD2と同一の働きをします．

D4 : このビットに"1"が立つとAフラグをリセットします．ただし，このビットの"1"は保持されず，Aフラグをリセットすると，クリアされます．

D5 : Bフラグをリセットします．リセットデータは保持されません．

D6, D7 : この2ビットは上記のD0～D5とは違った意味を持っており，チャネル3のモードを設定します．(表4.3)

表4.3　MODE設定

MODE	D7	D6	内　　　　　容
通常	0	0	チャネル3は通常の発音モードです．F-NumberはA2H・A6Hで指定されます．
効果音	0	1	チャネル3のF-Numberは各スロットごとに指定できます．1スロットはA9H・ADH, 2スロットはAAH, AEH, 3スロットはA8H・ACHのデータを使います．
音声合成	1	0	チャネル3のF-Numberは10の場合と同様ですが，MODEはCSMの音声合成モードとなり，チャネル3のKey-on/offはタイマAでコントロールされます．

(F-Numberについては(b−1)参照)

(a−3) Key-on/off : ADDRESS[28H]

このレジスタによって，FM12スロットのKey-on/offがコントロールされます．レジスタの下位2ビットがチャネルを指定します．そして上位4ビットが各チャネルのスロットを指定します．指定したチャネル・スロットに"1"を立てるとKey-onとなり，"0"を書き込むとKey-offとなります．

28H	D7	D6	D5	D4		D1	D0

チャネル割り当て

D1	D0	チャネル
0	0	チャネル1
0	1	チャネル2
1	0	チャネル3

スロット割り当て

D4	第1スロットのON/OFF
D5	第2スロットのON/OFF
D6	第3スロットのON/OFF
D7	第4スロットのON/OFF

例）

	D7	D6	D5	D4	D3	D2	D1	D0
チャネル1の全スロットをON	1	1	1	1	*	*	0	0
チャネル2の全スロットをOFF	0	0	0	0	*	*	0	1
チャネル3の1，2スロットのみON	0	0	1	1	*	*	1	0

*はDON'T CARE

(a—4) プリスケーラ機能：ADDRESS[2DH]・[2EH]・[2FH]

ここでは，FM・SSG各音源の周波数をコントロールします．この3つのアドレスに関しては，データビットはなく，アドレスを指定するだけです．分周数とアドレスの関係は表4.4のようになります．ただし，FM音源の出力はこの周波数の倍で出されます．

表4.4　入力クロックと内部クロックの関係

2DH	2EH	2FH	FM音源の分周数	SSG音源の分周数	OPNに入力できる最大周波数
—	—	A	1/2	1/1	1.4 MHz
A	—	—	1/6	1/4	4.2 MHz
A	A	—	1/3	1/2	2.1 MHz

（Aはそのアドレスを入力し，—は入力しないことを意味します.）

このプリスケーラの注意点として，初期設定時は，FM音源に対しては1/6，SSG音源に対しては1/4の分周数に設定されています．

(a—5) Detune/Multiple : ADDRESS[30H～3EH]

MultipleはF-NumberとDetuneで得られる位相情報に対して，表4.5で表されるような倍率の情報であり，DetuneはF-Numberにわずかの周波数ズレを与える情報です．

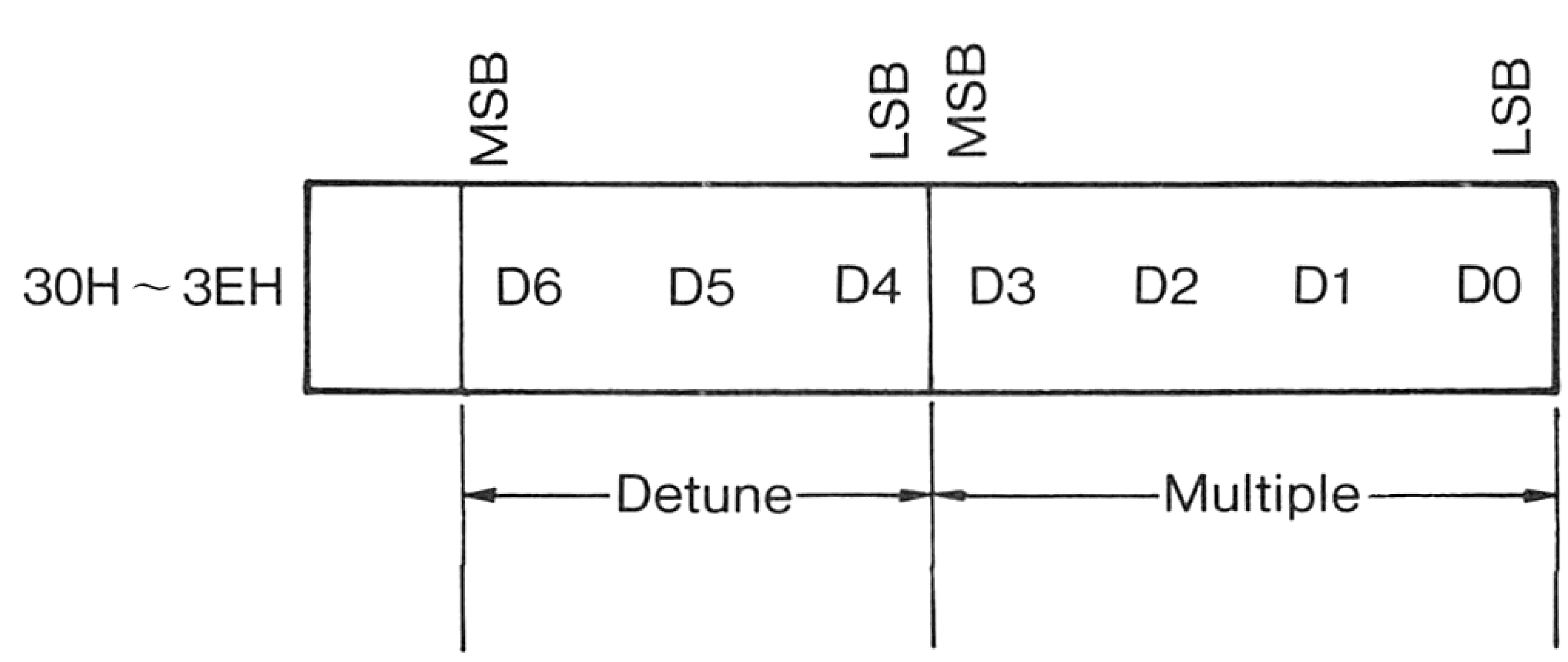


表4.5　倍率

Multiple情報	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
倍　　率	1/2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Detune/MultipleよりSSG-EGまでの7種類のFM制御情報は，各スロットごとに決めることができます．各アドレスとスロット・チャネルの対応は表4.6－aのとおりです．表からもわかるように，チャネルとアドレスの下位2ビットが対応しているため，アドレスの下位2ビットが“11”になることはありません．また，2スロットと3スロットの順序が逆になっています．

表4.6－a　アドレスとチャネル・スロットの関係

ADDRESS	チャネル	スロット	ADDRESS	チャネル	スロット
＊0H	1	1	＊8H	1	2
＊1H	2	1	＊9H	2	2
＊2H	3	1	＊AH	3	2
＊4H	1	3	＊CH	1	4
＊5H	2	3	＊DH	2	4
＊6H	3	3	＊EH	3	4

（＊は3より9までの値）

また，F-Number/BlockとSelf-Feedback/Connectionのデータはチャネルごとに決められるものであり，そのデータは，対応する4スロットに共通です．アドレスとチャネルの関係は次のとおりです．

表4.6－b　アドレスとチャネルの関係

ADDRESS	チャネル	ADDRESS	チャネル	ADDRESS	チャネル
＊0H，＊4H	1	＊1H，＊5H	2	＊2H，＊6H	3

（＊はAあるいはB）

(a—6) Total Level : ADDRESS[40H～4EH]

トータルレベルとは，エンベロープジェネレータの出力に対して，減衰量を加算し，変調度（音色）及び音量の制御をするために用いられます．減衰量は最小分解能を0.75dBとして，各ビットの重み付けは表4.7のとおりです．

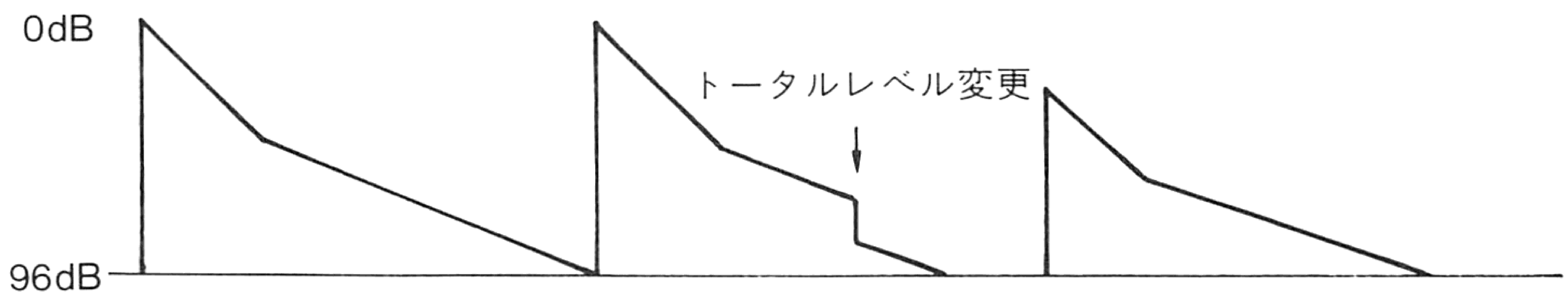
40H～41H	／	D6	D5	D4	D3	D2	D1	D0

表4.7 トータルレベルでの減衰量

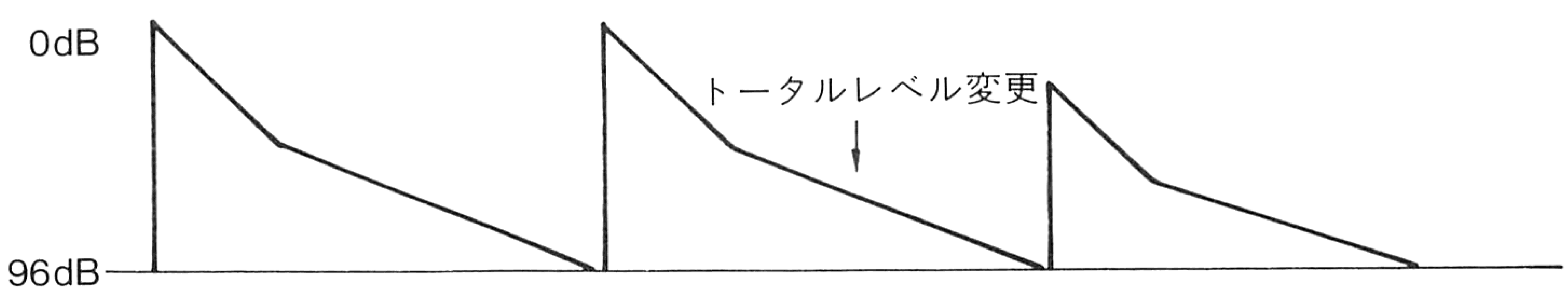
データビット	D6	D5	D4	D3	D2	D1	D0
減衰量(dB)	48	24	12	6	3	1.5	0.75

トータルレベルには，もう一つの働きがあります．それは，CSMのモードを選択した場合です．このときトータルレベルは3チャネルの音(CSM合成音)に対し，エンベロープのイニシャルレベルとなります．したがって，エンベロープの途中でトータルレベルを変えても，エンベロープの減衰量は変化せず，次のKey-on時のレベルが変わります．

・通常のエンベロープ



・CSMモードのエンベロープ



(a—7) Key Scale/Attack Rate : ADDRESS[50H～5EH]

Key Scaleはエンベロープの Rateを音程によって変化させるために設けられています．Key Scaling後のRateは次式によって表されます．式中のRは入力したRateであり，RksはKey Scaling量でこの値は表4.8のとおりです．

Rate＝2＊R＋Rks

(ただし，R＝0の場合は，Rate＝0とする.)

表4.8　RateのKey Scale

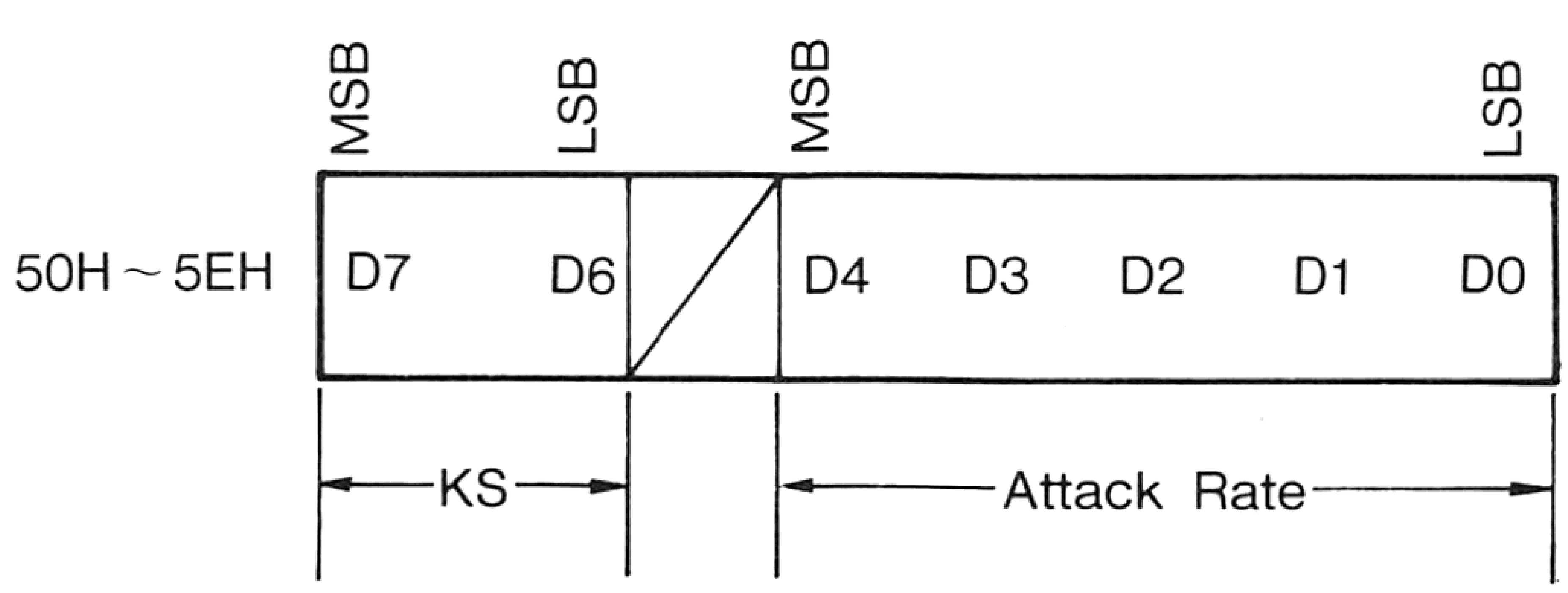
KS＼Key-Code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
2	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

KS＼Key-Code	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3
1	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7
2	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15
3	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Key-Codeについては(b−1)を参照

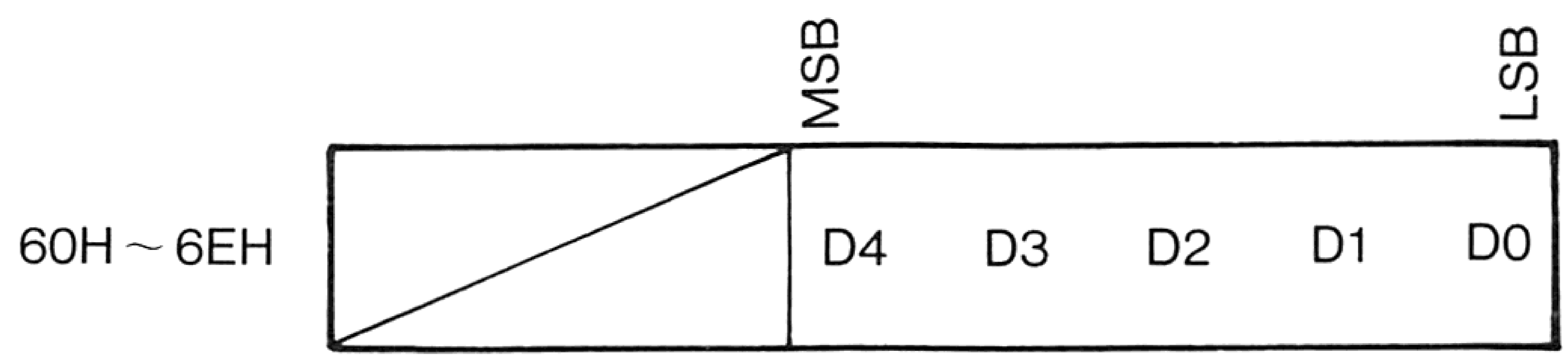
Attack Rateは音の立ち上がりの時間を設定します.

なお，エンベロープの形状は資料4.3.3 (e)のエンベロープ・ジェネレータの項を参照してください.



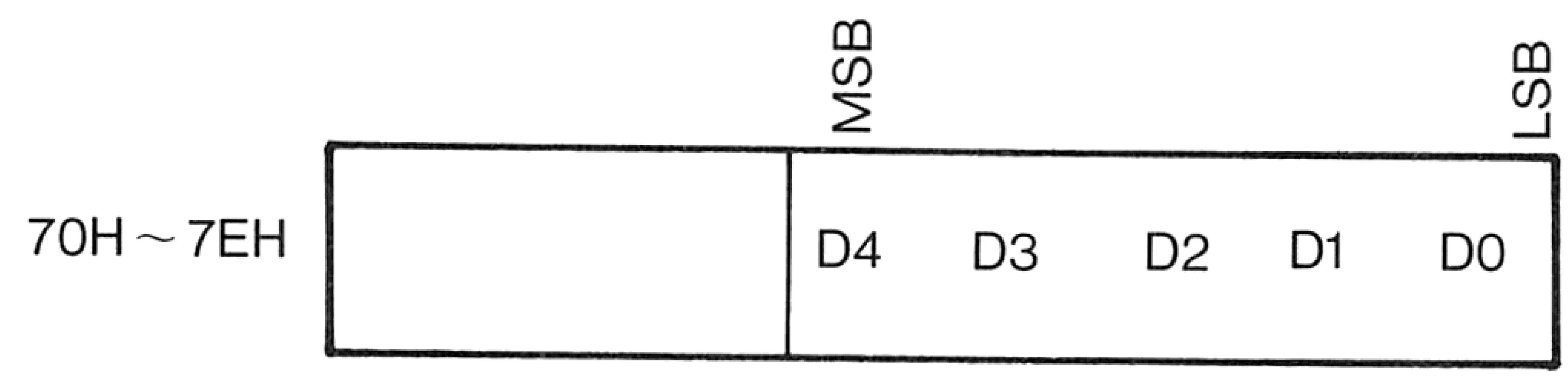
(a−8) Decay Rate : ADDRESS[60H～6EH]

Decay時のRateを設定します.



(a—9) Sustain Rate : ADDRESS[70H～7EH]

Sustain時のRateを与えます.



(a—10) Sustain Level/Release Rate : ADDRESS[80H～8EH]

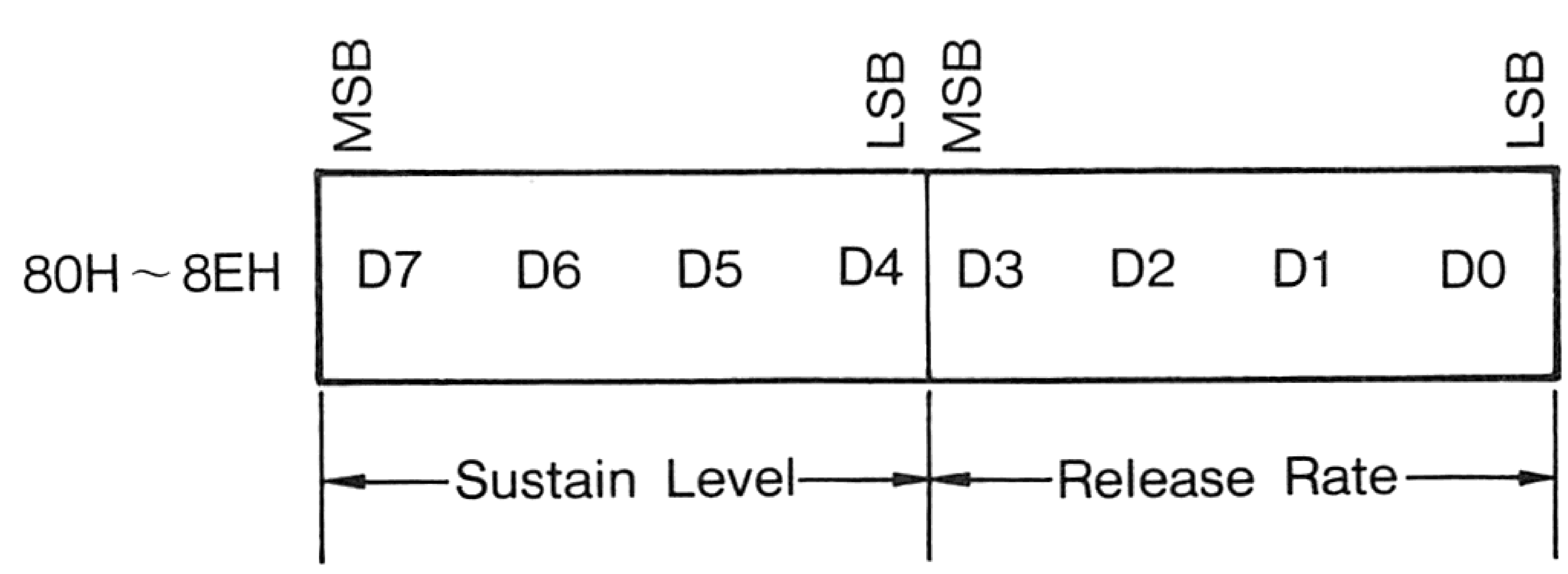
Sustain LevelはDecay Rateと, Sustain Rateの切り替え点を与えます. その重み付けは表4.9で表され, 値は減衰量を示します.

表4.9　サスティーン・レベルと重み付け

データビット	D7	D6	D5	D4
減衰量(dB)	24	12	6	3

(ただし, D4～D7がすべて"1"の場合は, 93dBとなります.)

Release RateはKey-offのときのRateを与えます. ただし, Release RateはLSBが"1"に固定されているため, 他のRateと異なって4ビットしか入力できません.



(a—11) SSG-Type Envelop Control : ADDRESS[90H～9EH]

SSG-Type Envelopとは, 後述のSSG音源で作られるエンベロープと同一形状を与える機能です. この4ビットのデータ中, D3がこのタイプのエンベロープをとるか否かのスイッチとなっており, 残り3ビットがデコードされ, 図4.30の形状のエンベロープを発生します.

このエンベロープが使われた場合の注意点は, Rateの使い方が通常の場合とは異なっていることです. このモードでは, パラメータはDecay, Sustain, Releaseの各Rateです. この中でRelease RateはKey-off時のみ有効であるのは, 通常モードと変わりありません. したがって, Key-on中に音量が増加したり, 減少したりする場合の制御は,

Decay及びSustainの2つのRateでなされます.(音量の増加は減少の場合を逆にたどっていく形になります.)もう一つ通常モードと異なる点は減衰時間の違いです. 通常モードより変化が速くなります.

90H～9EH		D3	D2	D1	D0

＊ Attack Rateは1FHに固定されていなければなりません.

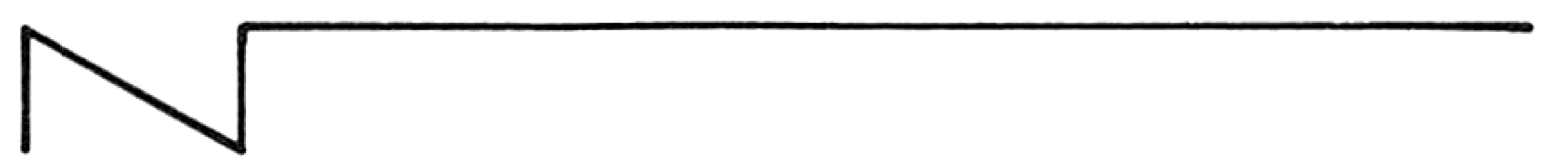
D3 D2 D1 D0	エンベロープ波形
1 0 0 0	
1 0 0 1	
1 0 1 0	
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

図4.30 SSG-Typeのエンベロープ

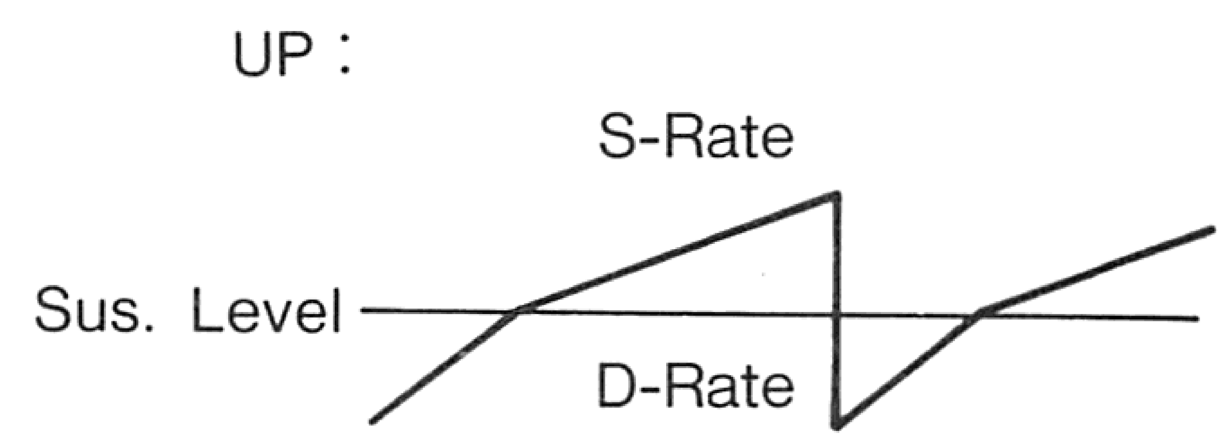
このエンベロープはエンベロープジェネレータ出力の波形であり, 対数出力です. OPNの出力ではリニアに変換されますので増減の傾きは変化します.

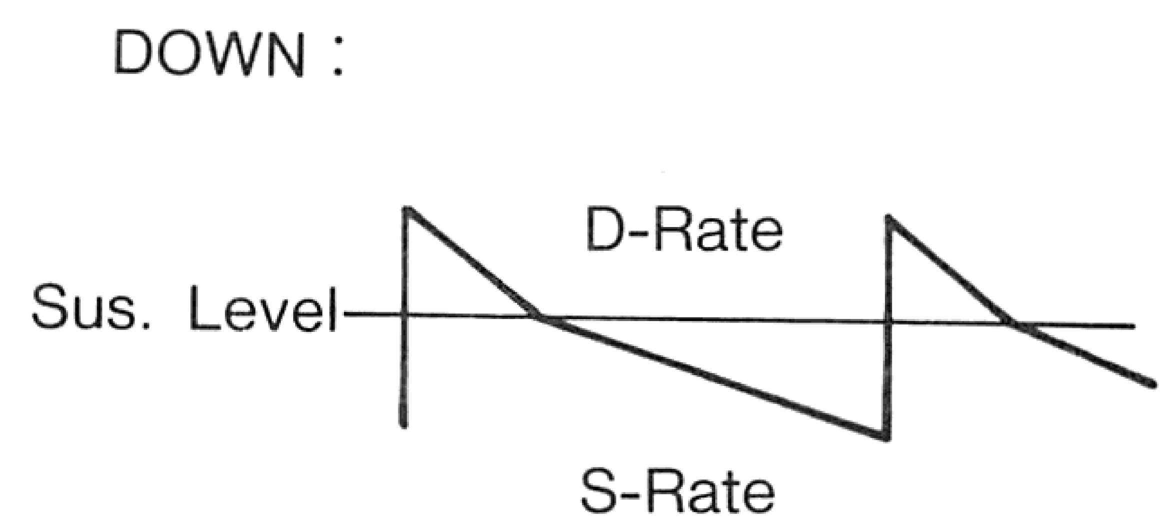
SSG-Type EGを使ったときの注意点としては, 次の2つが挙げられます.

(1) Attack Rateは通常1FHにしなければなりませんが, 08H, 09H, 0BHなどは普通のエンベロープ時のRateで変化します. その他のものについては十分な制御はできません.

(2) Decay Rate, Sustain Rate, Sustain Levelはエンベロープが, UP時又は,

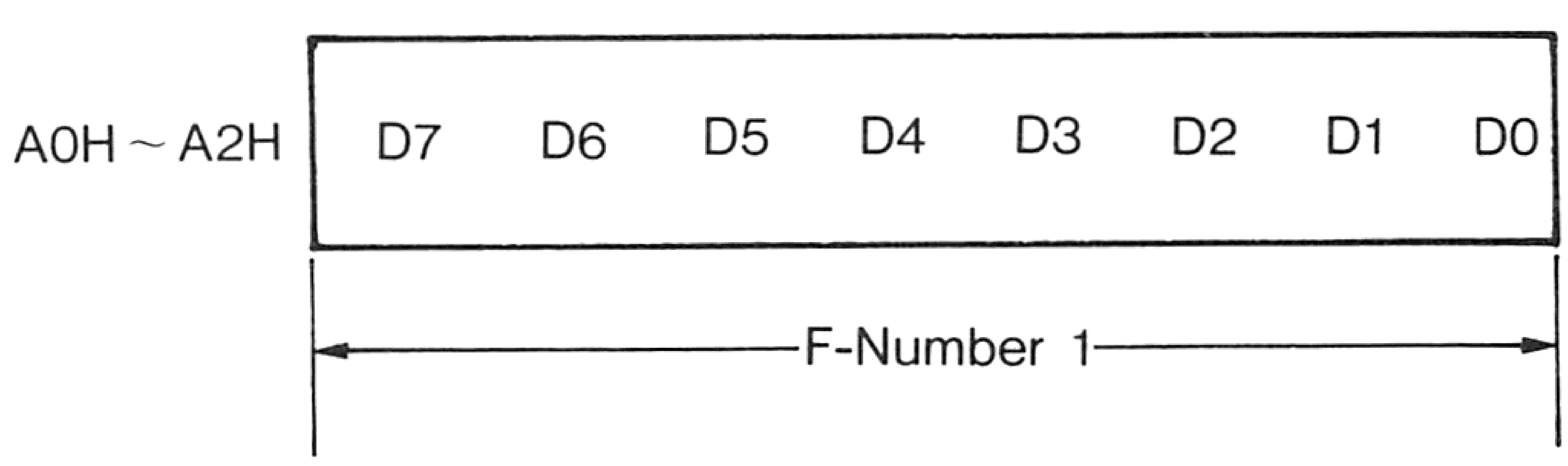
DOWN時では異なります．

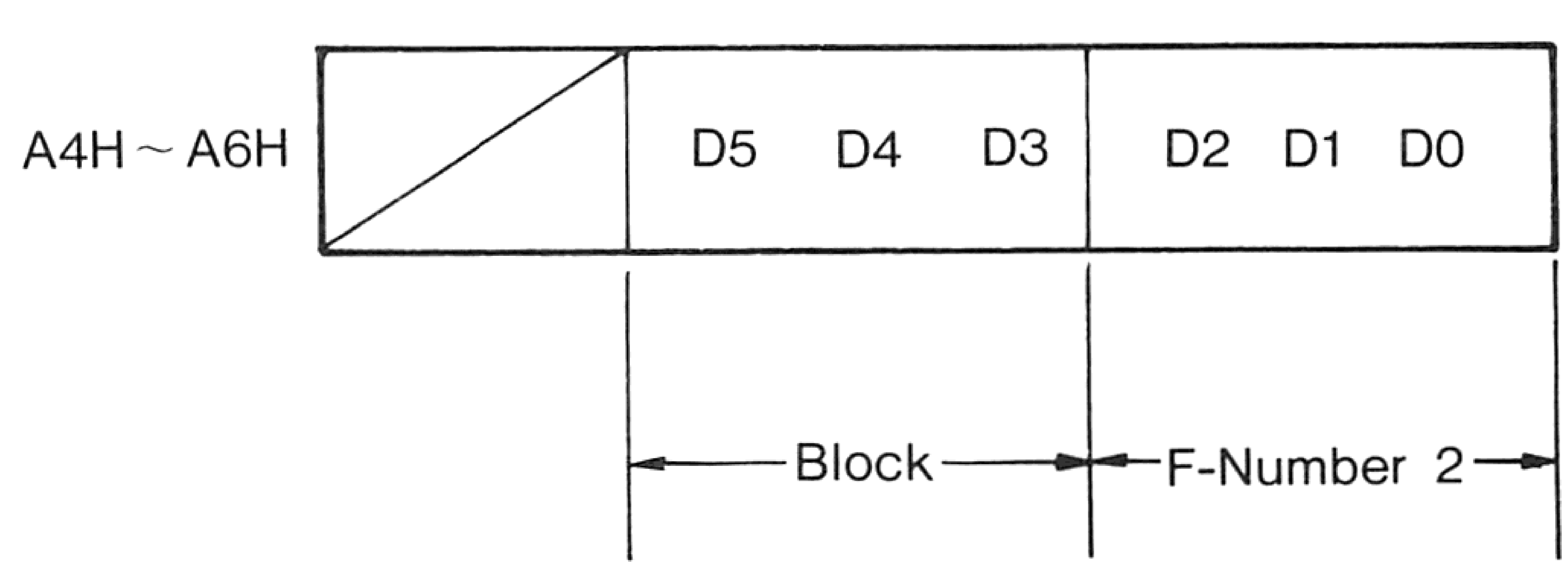




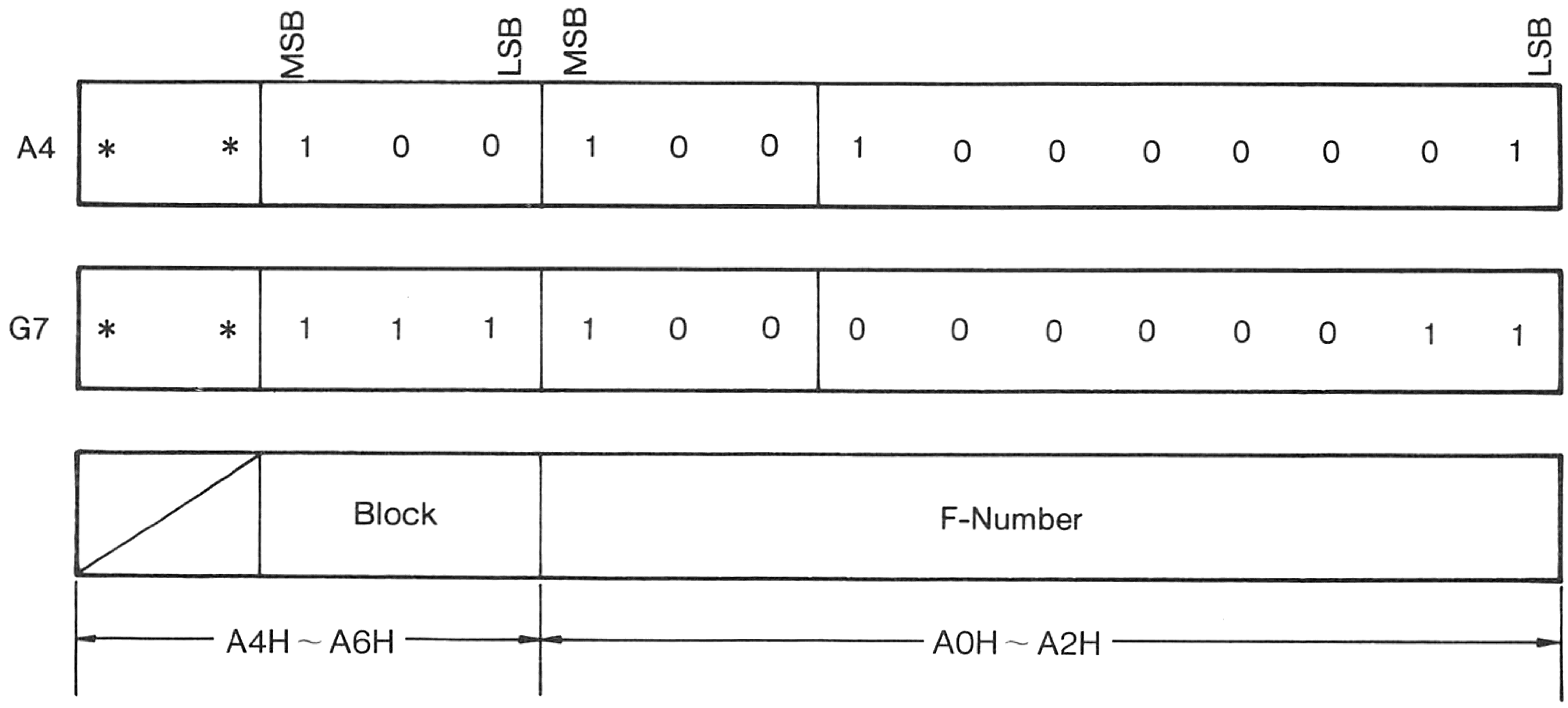
(a—12) F-NumberとBlockデータ：ADDRESS[A0H～A2H, A4H～A6H]

このアドレスでF-NumberとBlockのデータがセットされます．データをセットする場合の注意点としては，まず最初にBlockとF-Numberの上位3ビット(F-Number2)をセットし，次にF-Numberの下位8ビット(F-Number1)を入力します．F-NumberとBlockの内部のレジスタへのローディングはF-Number1がセットされたときのみ動作しますから，データの入力の順序が異なりますと，BlockとF-Number2のデータは違ったデータとなります．





例）　音程データ(@øM＝1.2MHz)



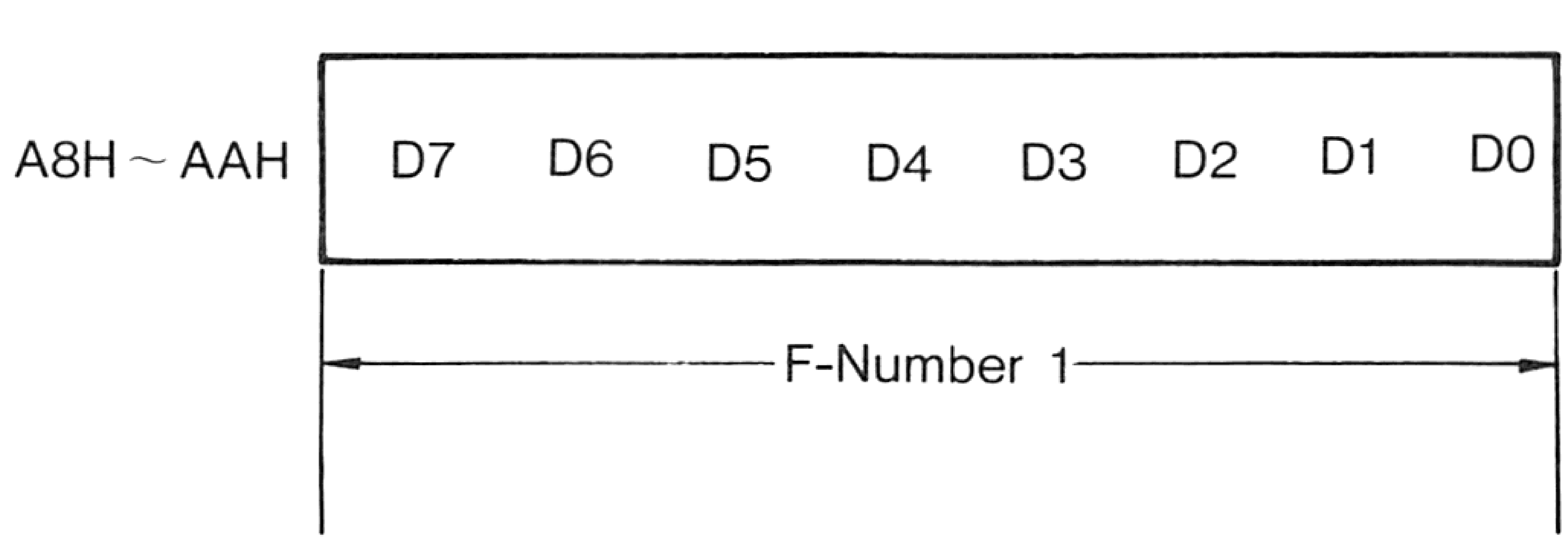
(a—13) 3チャネルのF-NumberとBlock：ADDRESS[A8H～AAH，ACH～AEH]

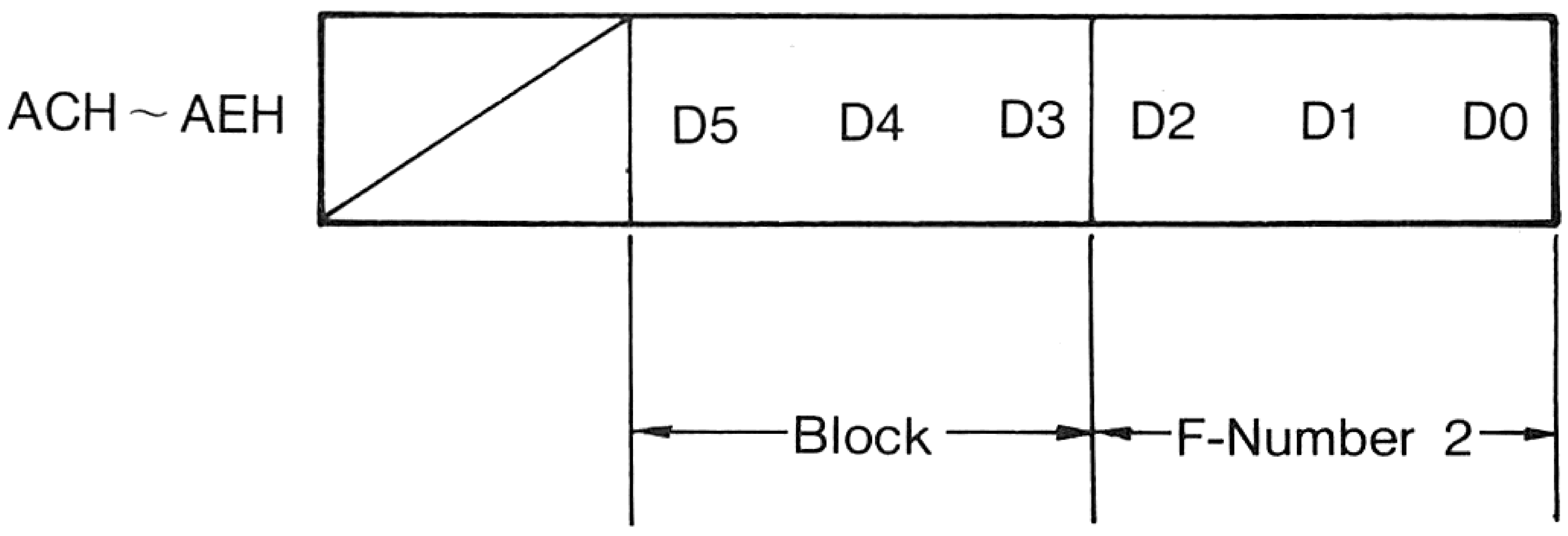
このアドレスでセットされるF-NumberとBlockは効果音モードや音声合成モードで使われるデータです．データセットの方法は(a—12)で述べた方法と同様で先にBlock，F-Number2のデータを，次にF-Number1のデータをセットします．

ここでセットしたデータとスロットの関係は次のようになります．

スロット番号	F-Number/Blockのアドレス
1	A9H, ADH
2	AAH, AEH
3	A8H, ACH
4	A2H, A6H

上記のスロットとアドレスとの関係は効果音モードと音声合成モードのときに適用されるのであって，通常モードでは全スロットともA2HとA6Hのアドレスで与えられます．





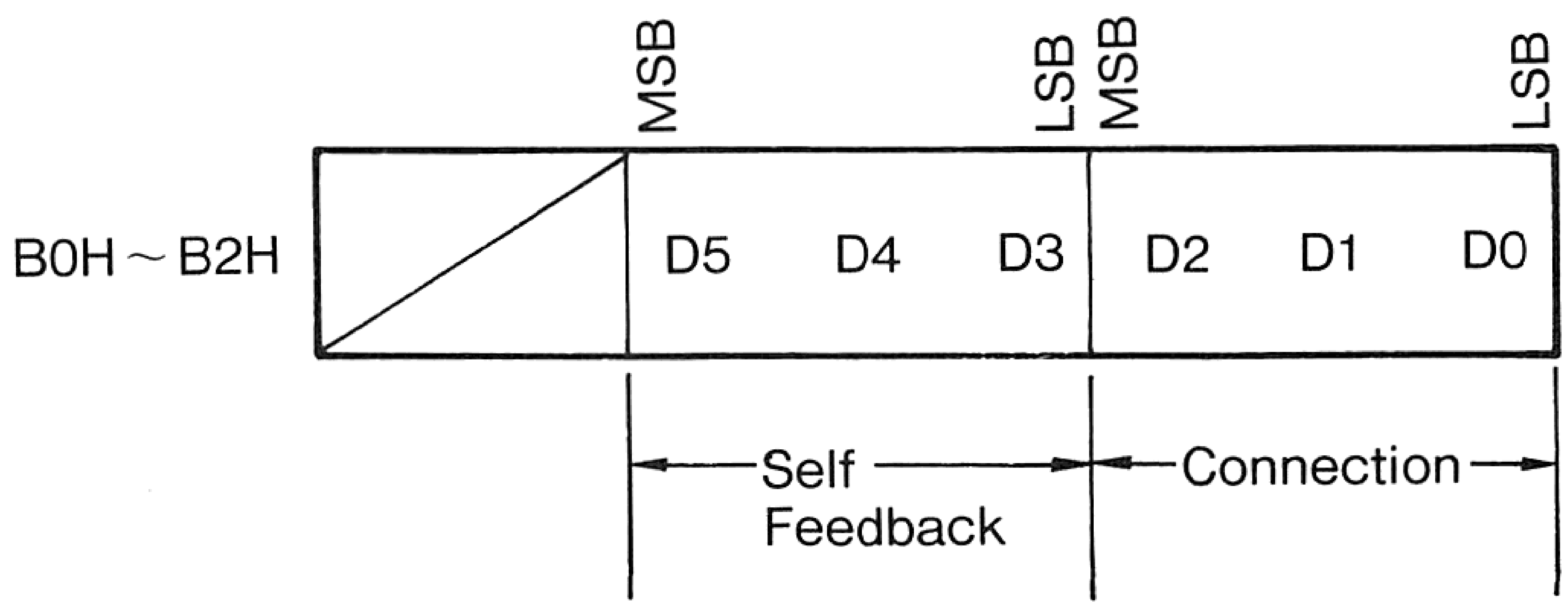
(a—14) Self-Feedback/Connection : ADDRESS[B0H～B2H]

Self-Feedbackというのは，各チャネルの第1スロットの変調度を決めるデータです．つまり，第1スロットは自分自身の出力を変調信号としているため，その変調度合をこのデータで制御します．変調度は表4.10に示すとおりです．

表4.10　変調度

D3～D5	0	1	2	3	4	5	6	7
変調度	OFF	$\pi/16$	$\pi/8$	$\pi/4$	$\pi/2$	π	2π	4π

Connectionは4つのスロットをどのように変調波と被変調波に分け，どういうふうに組み合わせるかを指定するデータです．それらの組み合わせは全部で8つあり，その構成については，2章　VOICE文のアルゴリズム表を参照してください．



(b)　フェイズ・ジェネレータ(PG)

フェイズ・ジェネレータは必要な周波数に応じた増分を単位時間(1 DACサイクル)ごとにアキュームレートして位相値を得る回路です．この増分はレジスタから送られてくる周波数情報(Block/F-Number/Detune/Multiple)から作られます．4つある周波数情報の中で柱となるのは，F-Numberであり，これにより1オクターブ中の音程が決定されます．そこで，まずF-Numberから説明します．

(b—1) F-Number/Block

位相の増分は，発音したい周波数と，入力クロック周波数(サンプリング時間)が決ま

ると，すぐに求めることができます．これは次式になります．

$\triangle P = f\ Mus * 2^{20} / f\ SAM \quad f\ SAM = fM/(12 * n) \quad n = 2, 3, 6$

f Mus：発音したい周波数

f SAM：サンプリング周波数

f M：入力クロック周波数

n：プリスケーラコントロールで得られる定数

〔(a−4)参照〕

これで位相の増分は求まりますが，この増分を管理するのはビット数が多くて大変です．そこで，増分は1オクターブ分のデータのみを管理できるビット数に制限し，指定したオクターブごとにその増分をシフトすることにより，位相を求めることとします．これにより，オクターブ情報の指定をBlockデータに，増分をF-Numberに入力すれば全音域をカバーできることになります．以上の操作とF-Numberを11ビットで表現することにしたため，結局F-Numberは次式で求められます．

$F = (f\ Mus * 2^{20} / f\ SAM) / 2^{b-1}$ （bはオクターブデータ）

例） 3.6MHzの入力クロックでA4(440Hz)のF-Numberを求める．

$F\ A4 = (440 * 2/50000)/2^{4-1} = 1153$ （誤差−0.65セント）

次にKey Scaleを行うときに問題となる1オクターブ内の分割について少し説明します．

Key Scaleは通常1オクターブ内を4分割（3音ずつの組）して行うことが多くありますが，F-Numberを用いると一定のセントで分割することは困難となります．そこで，OPNではF-Numberの上位4ビットをデータとして分割を行っています．ただし，この方法では入力周波数によって分割が異なります．

1オクターブ内の周波数分割

$N4 = F11$

$N3 = F11 \cdot (F10 + F9 + F8) + \overline{F11} \cdot F10 \cdot F9 \cdot F8$

以上の操作で得たN4, N3の2ビットとオクターブデータの3ビットの計5ビットでKey Scaleを行います．

	Block	Note
	B3　B2　B1	N4　N3
Block, Noteを分離した重み付け	D2　D1　D0	D1　D0
Block, Noteをまとめた場合の重み付け	D4　D3　D2	D1　D0

(b—2) Detune/Multiple

レジスタの項でも説明したように，Detuneは与えられたF-Numberに対して周波数変位を与えるために，周波数に応じた定数を加算あるいは減算します．またMultipleはDetuneの操作を受けたF-Numberに指定された倍率を乗ずる働きをします．

(b—3) アキュームレータ

Detune及びMultipleの回路を通ったF-Numberを累積加算し，その結果をオペレータに送ります．このアキュームレータは加算のみを行い，オーバーフローが生じてても無視します．これは，オーバーフローが起こったことは弧度法でいう2πを通過したにすぎないからです．

資料4.3.3　SSG音源部

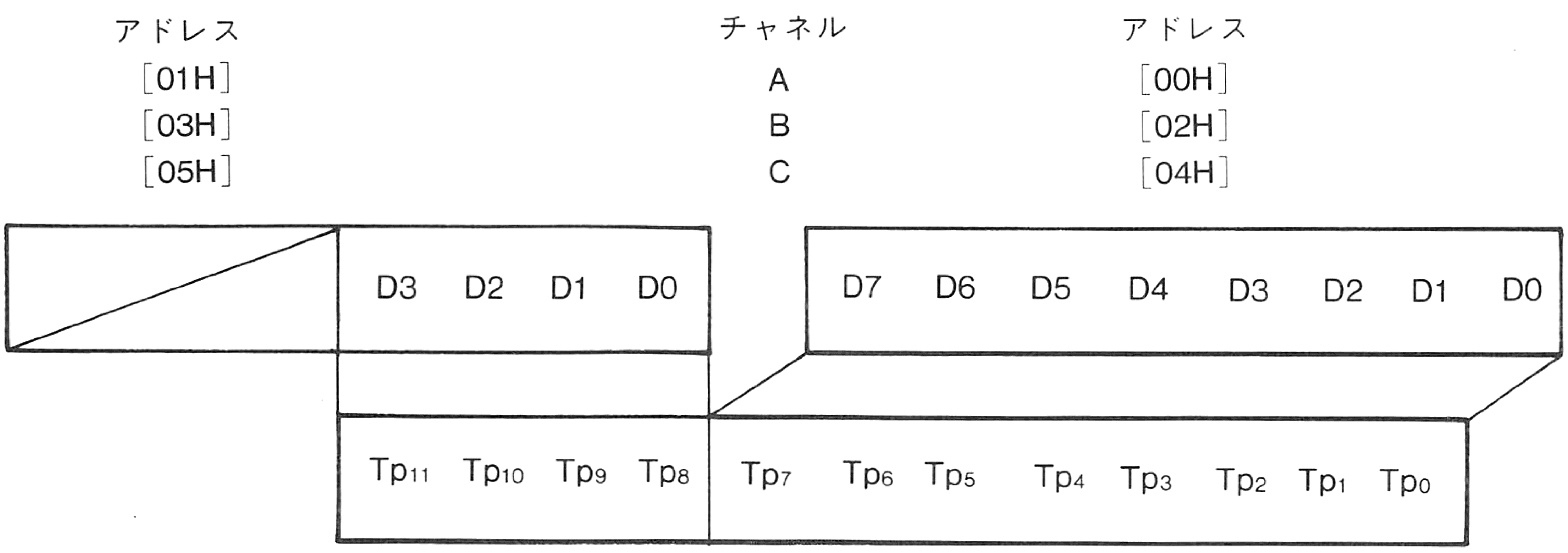
SSG音源部はDACを内蔵した音源であり，そのレジスタアレーにデータを書き込むだけで楽音を発生します．SSG音源内部のレジスタアレーはFM音源部と異なり，READ/WRITEが可能です．以下，各レジスタごとにその機能を説明していきます．

(a)　トーン・ジェネレータ・コントロール：ADDRESS[00H～05H]

SSG音源部もFM音源部と同様に3チャネルの楽音を発生することができます．この3チャネルの楽音の周波数を設定するのがこのレジスタです．楽音周波数は12ビットで構成されるため，1チャネルにつき2つのアドレスを必要とし，そのビットの対応は下記のとおりです．また，発信周波数は次式で求められます．なお，出力波形は1：1の矩形波です．

$$f_{tone}=f_m/(8*n*T_p) \qquad n=1, 2, 4$$

fm：入力クロック周波数
n：プリスケーラ・コントロールで定められる定数
Tp：発信周波数設定値



注意：この分周においてTp=0とTp=1は同一値となります．

(b) ノイズ・ジェネレータ・コントロール：ADDRESS[06H]

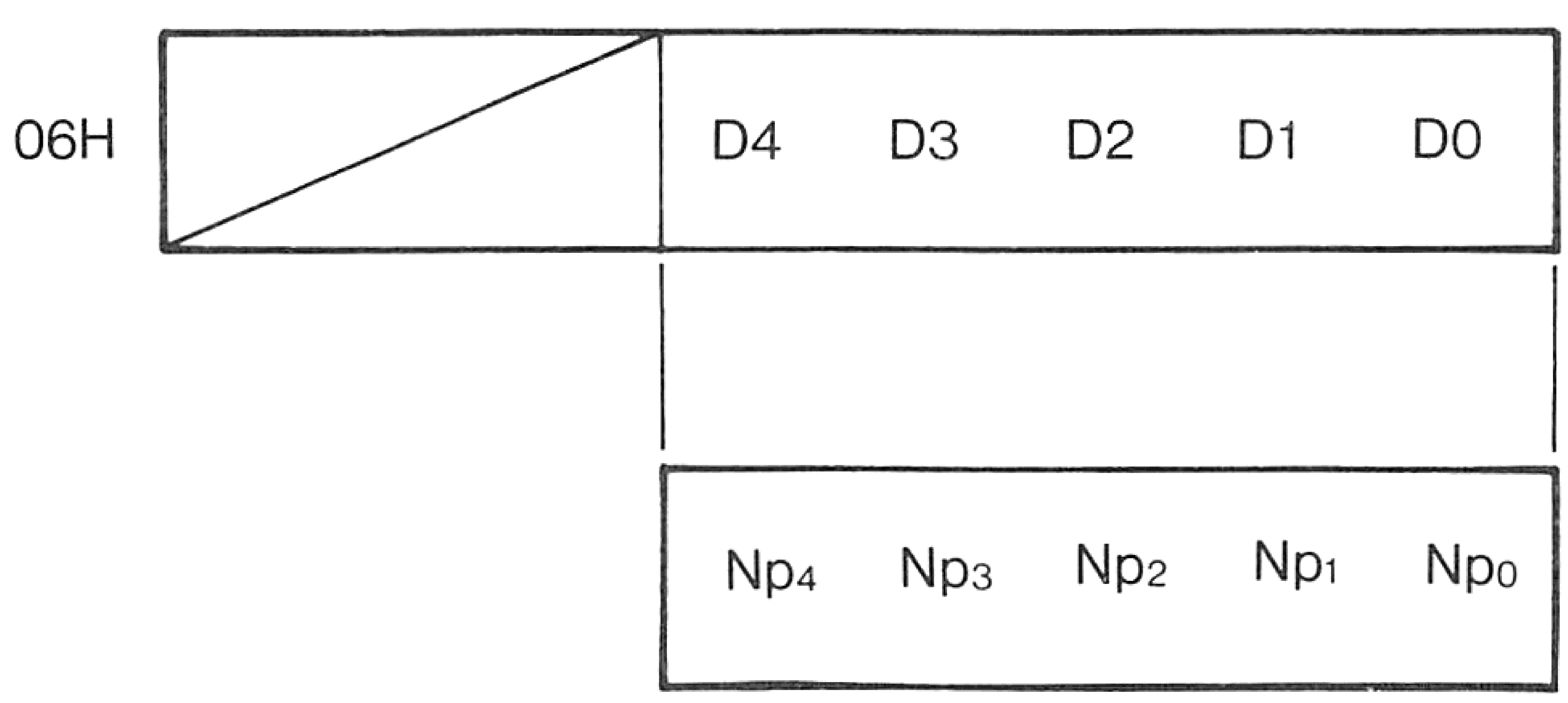
ノイズ源は系列長の疑似ランダムノイズです．そして，この系列の発生をコントロールするのがこのレジスタの働きです．系列発生周期(ノイズ周波数)はレジスタの下位5ビットで決められ，次式で求めることができます．

$$fN = fM/(8 * n * Np) \qquad n=1, 2, 4$$

fM ：入力クロック周波数

n ：プリスケーラ・コントロール設置値

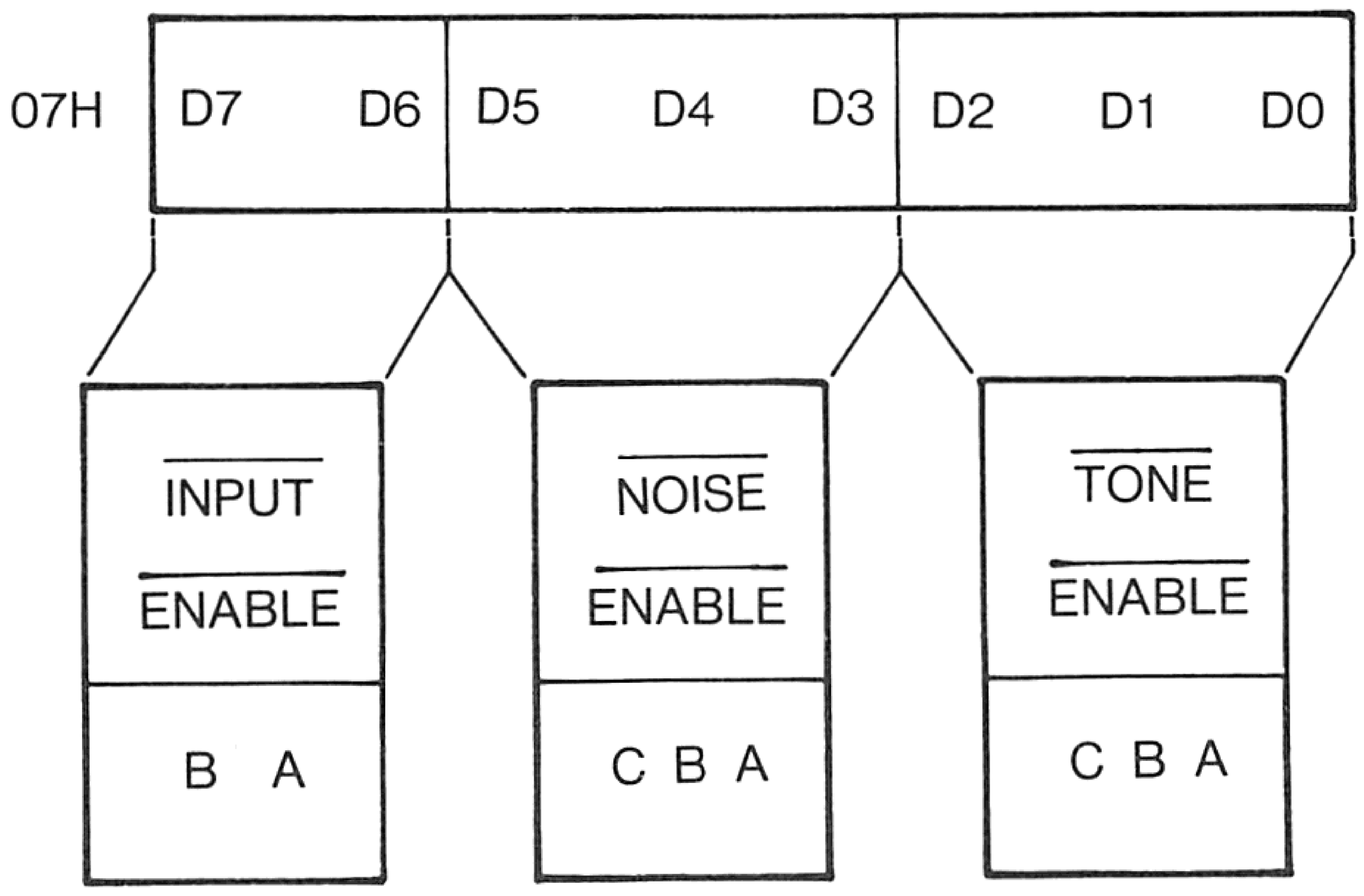
Tp ：ノイズ周波数設定置



注意：この分周においてNp=0とNp=1は同一値となります．

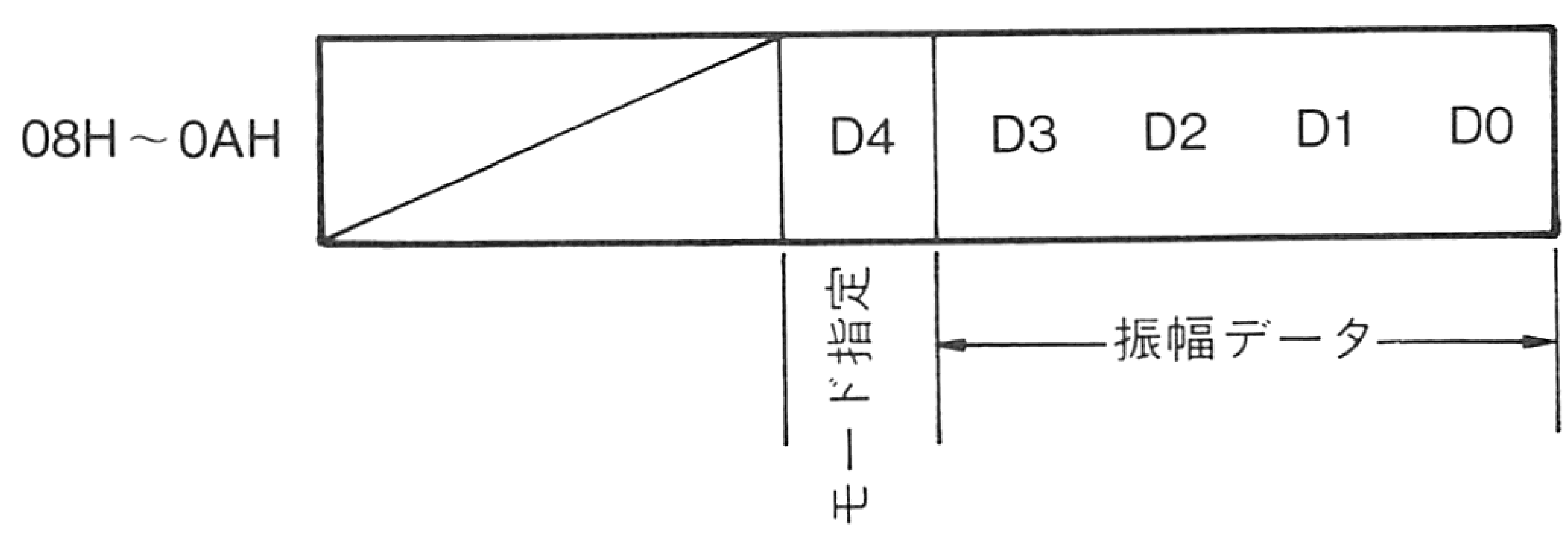
(c) ミキサー・I/Oコントロール：ADDRESS[07H]

このレジスタは3つのノイズ/トーン・ミキサーと2つの汎用I/Oポートのコントロールをします．ミキサーは3つのチャネル毎に，ノイズとトーンともに出力するのか，いずれか一方だけを出力するのか，あるいは両方とも出力しないのかをビット0～5で制御します．(データ"0"で発音します.) また，2つのI/Oポートの入出力制御はビット6と7でなされます(データ"0"でINPUTです)．



(d)　**振幅コントロール：ADDRESS[08H～0AH]**

A, B, C 3つのチャネルに対応するD/Aコンバータの振幅は，このレジスタによって決められます．レジスタのビット4が振幅のモードを決めるデータとなっており，"0"のとき固定振幅モードとなり，残りの4ビットで決められる値をとります．また"1"の場合は可変振幅モードとなり，後述するエンベロープ・コントロールの出力で決まる振幅となります．なお，D/Aコンバータは，入力5ビットの能力を持っており，固定振幅モード時の4ビットの振幅データは，D/Aコンバータの入力上位4ビットに対応します．



(e)　**エンベロープ・ジェネレータ・コントロール：ADDRESS[0BH～0DH]**

変化のあるエンベロープを発生するために，2つのエンベロープ・コントロールがあります．その1つは，アドレス0BHと0CHで定められるエンベロープ周波数を制御することであり，もう一つはアドレス0DHのエンベロープの形状を制御することです．

まず，エンベロープ周波数について説明します．指定されたアドレスの2バイト16ビットのデータで入力周波数を分周することにより，エンベロープ周波数は求まります．それは次式で与えられます．

$fE=fM/(128*n*Ep)$　　$n=1, 2, 4$　　$Ep=256*CT+FT$

fM：入力クロック周波数

n：プリスケーラ・コントロール設定値

CT：0CHで与えられるエンベロープ周期設定値

FT：0BHで与えられるエンベロープ周期設定値

このエンベロープを作るカウンタの上位5ビットがD/Aコンバータへ送られ，振幅データとなります．

0BH							
D7	D6	D5	D4	D3	D2	D1	D0
F_{T7}	F_{T6}	F_{T5}	F_{T4}	F_{T3}	F_{T2}	F_{T1}	F_{T0}

0CH							
D7	D6	D5	D4	D3	D2	D1	D0
C_{T7}	C_{T6}	C_{T5}	C_{T4}	C_{T3}	C_{T2}	C_{T1}	C_{T0}

MSB															LSB
C_{T7}	C_{T6}	C_{T5}	C_{T4}	C_{T3}	C_{T2}	C_{T1}	C_{T0}	F_{T7}	F_{T6}	F_{T5}	F_{T4}	F_{T3}	F_{T2}	F_{T1}	F_{T0}

次にエンベロープの形状について説明します．先に述べたようにエンベロープはエンベロープ・カウンタの上位5ビットで決まります．したがってここでコントロールするのは，この上位5ビットのカウントの方法ということになります．アドレス0DHの下位4ビットがこれを決定します．各ビットの機能は次のように定義され，それらの組み合わせにより，図4.31－aで与えられるエンベロープの形状が得られます．

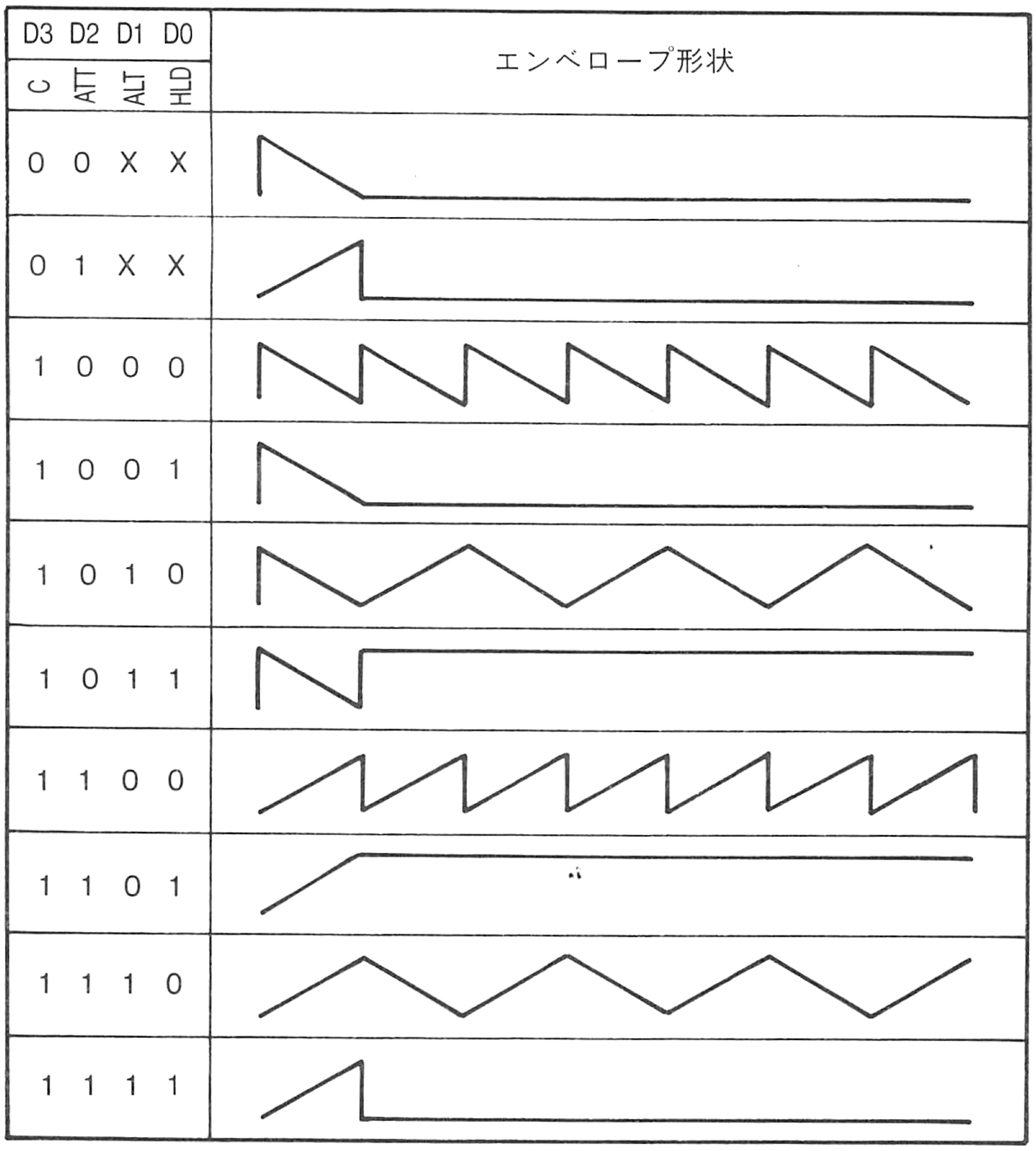
0DH		D3	D2	D1	D0
		C	ATT	ALT	HLD

C："1"にセットされたときはHLDのデータで定められる状態になり，"0"の場合は，エンベロープ・ジェネレータは1サイクル経過後すべて"0"にリセットされます．

ATT："1"にセットされたときはエンベロープ・ジェネレータはすべて"0"からすべて"1"へカウント・アップし，"0"であればすべて"1"からすべて"0"へカウント・ダウンします．

ALT："1"にセットされたときは，各サイクルごとにエンベロープ・カウントの方向を逆転します．

HOLD："1"のときは，エンベロープ・カウンタを1サイクル・カウントした後，カウンタの最終値をホールドします．ただし，ALTも"1"のときは，直前のサイクルの初期カウント値にリセットされます．



X：Don't care

図4.31－a　SSG部のエンベロープ波形

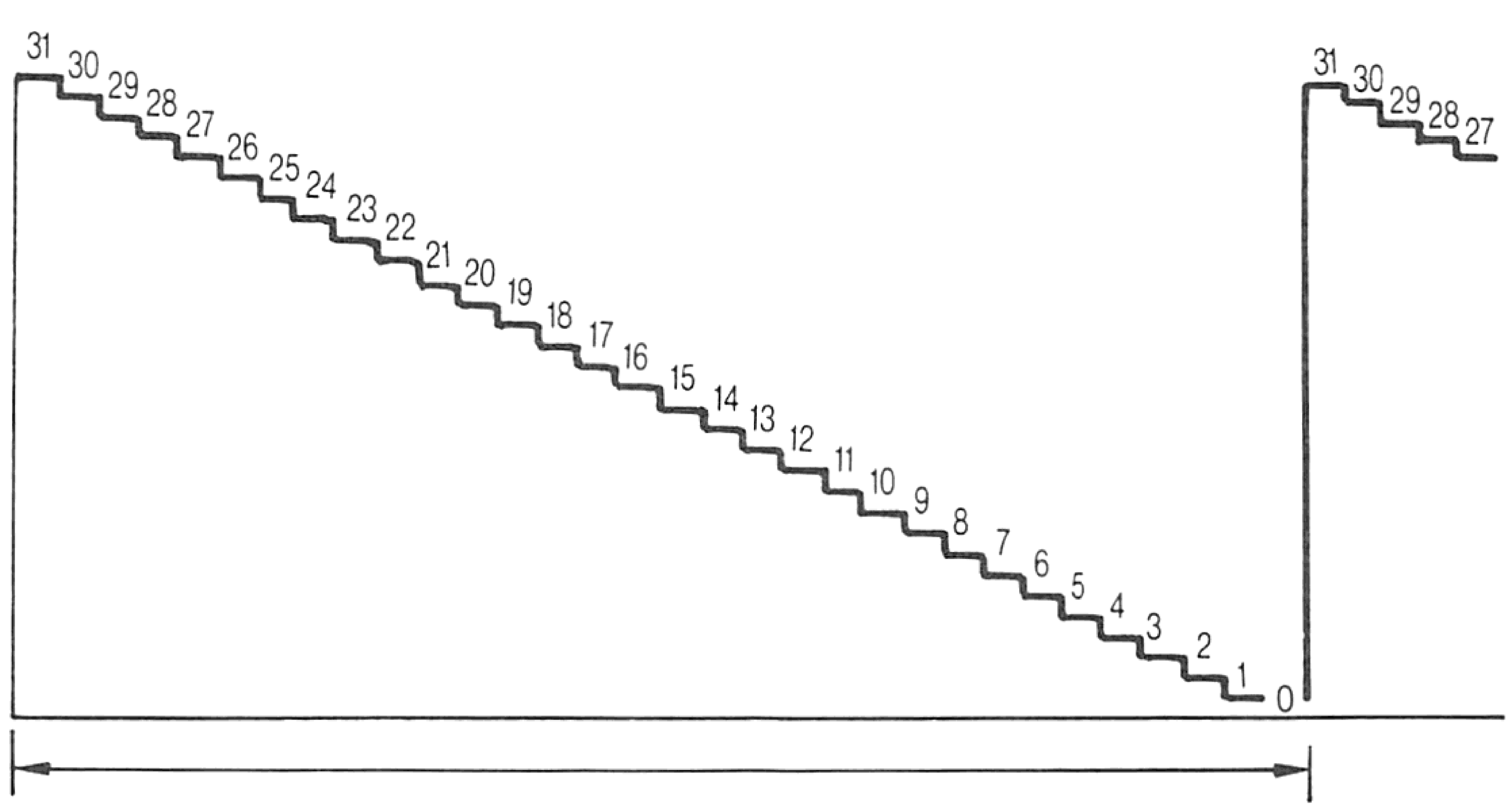


図4.31－b　1000データの詳細

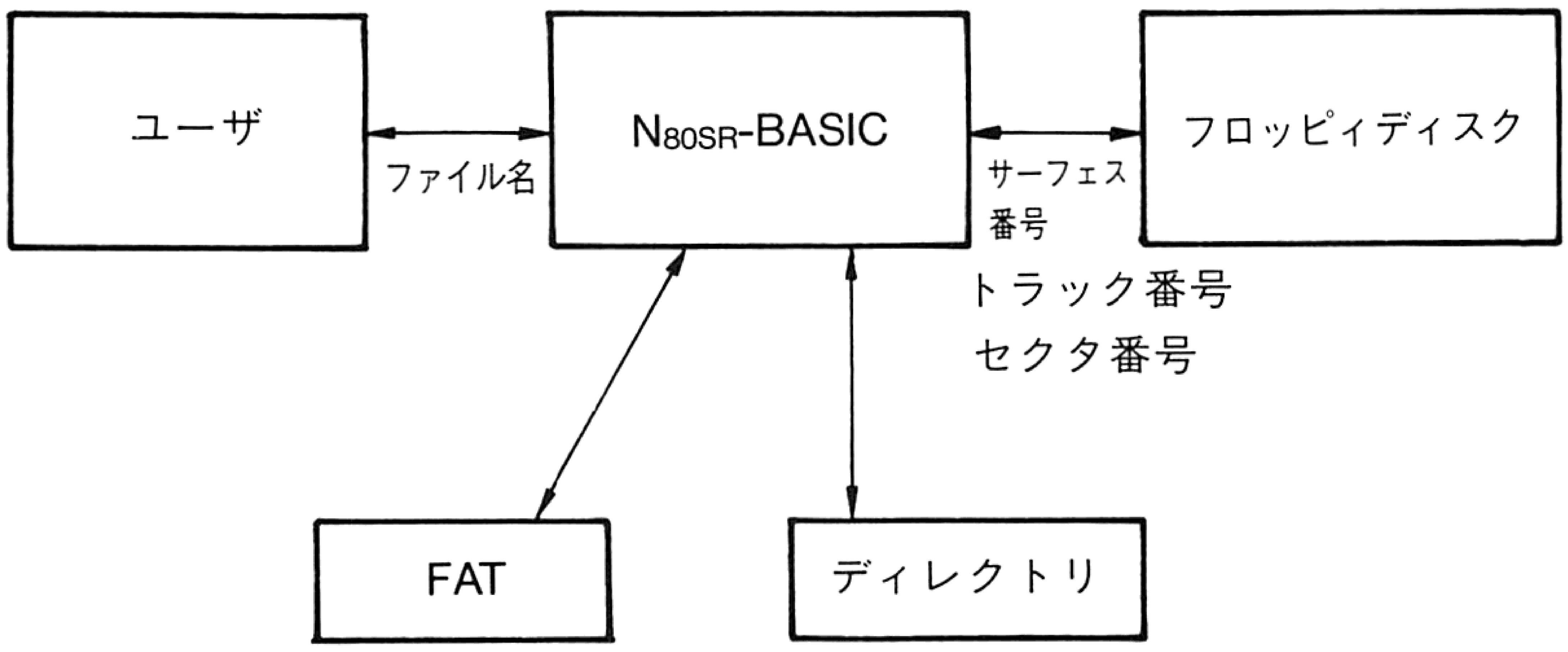
参考　資料4.1は，「デジタルキーボードハンドブック」（「日本楽器製造株式会社」発行）から

引用しています．また，資料4.3.1～4.3.3は，「YM-2203アプリケーションマニュアル」(「日本楽器製造株式会社」発行)から引用しています．

資料5　ディスクファイルの構造

N_{80SR}-BASIC DISKバージョンにおけるフロッピィディスクのファイル構造について解説します．

N_{80SR}-BASIC DISKバージョンでディスクファイルを扱うときは，すべてファイル名で行います．したがって，フロッピィディスクを直接管理するサーフェス番号，トラック番号，セクタ番号を使う必要はほとんどありません．なぜならば，ファイル名を実際のフロッピィディスクのサーフェス番号，トラック番号，セクタ番号に変換する仕事はN_{80SR}-BASICが自動的に行ってくれるからです．



したがって，BASICでファイルを扱われる方は以下の説明を読む必要はありません．DSKI$関数，DSKO$文や機械語モニタを使う必要のある方のみお読みください．

資料5.1　フロッピィディスク

N_{80SR}-BASIC DISKバージョンで使用できるフロッピィディスクは次の3種類です．

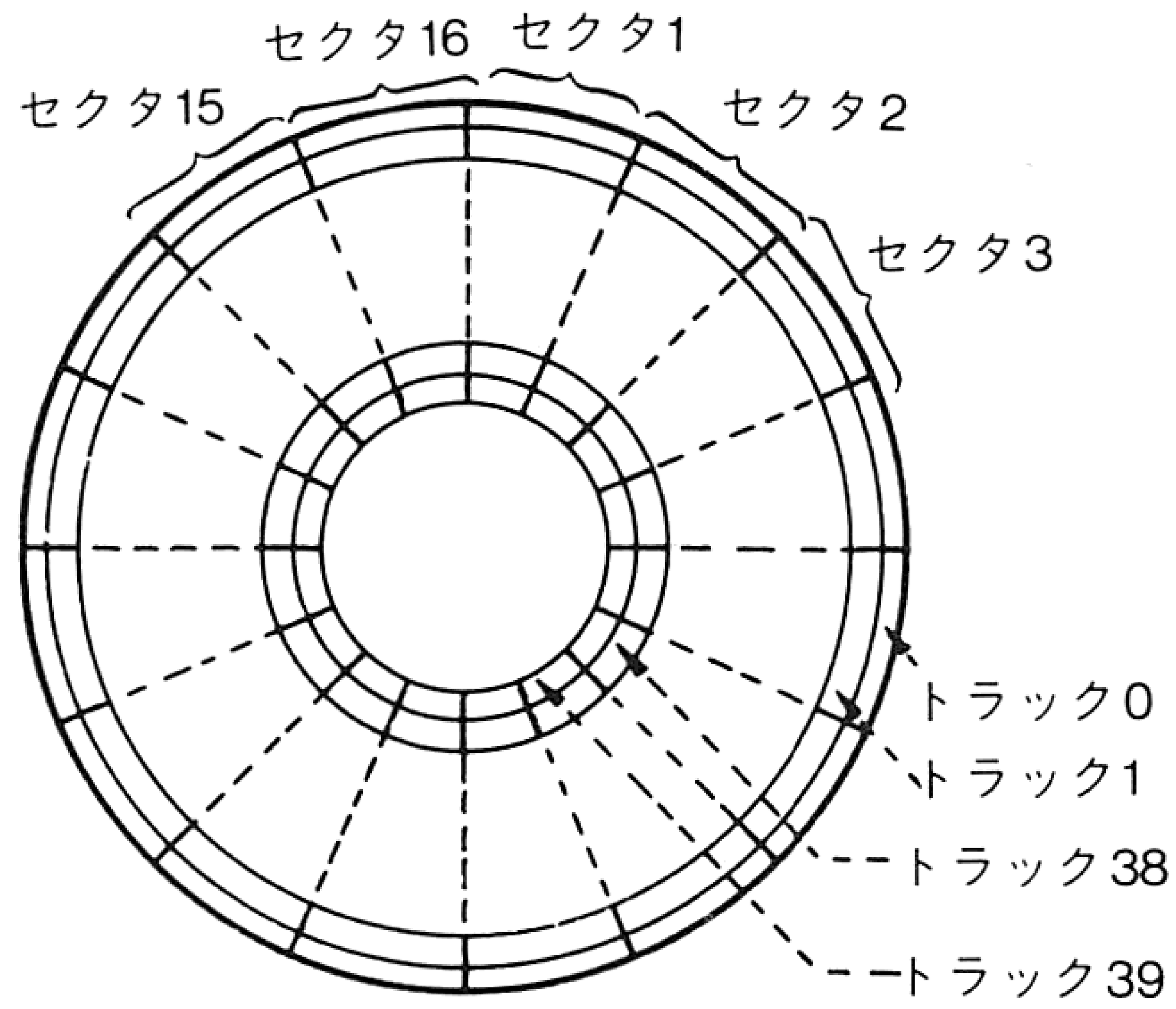
(1)　5.25インチミニフロッピィディスク(両面倍密度)

(2)　5.25インチミニフロッピィディスク(片面倍密度)

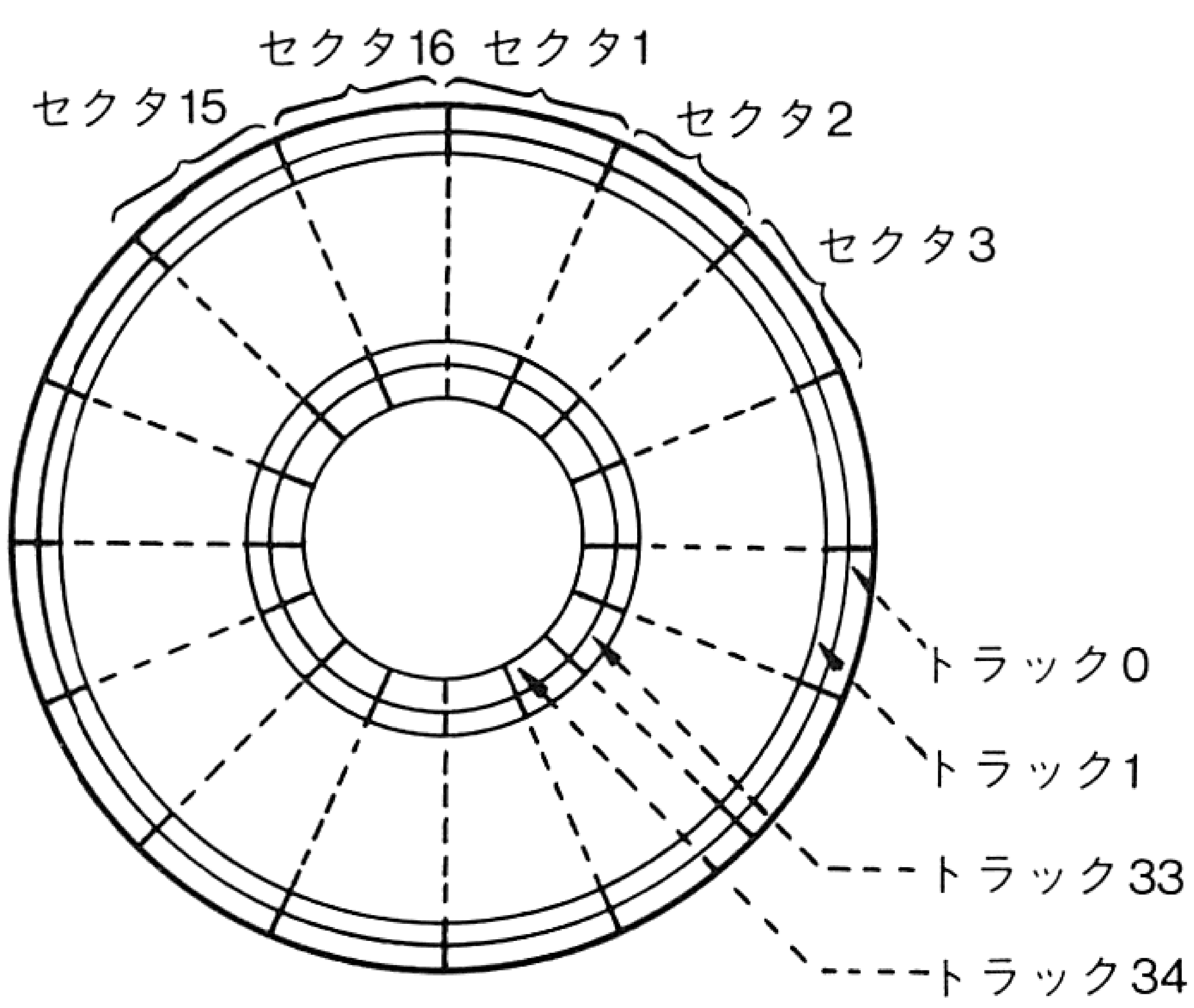
(3)　8インチフロッピィディスク(両面倍密度)

フロッピィディスクにはメモリと同じように番地が付いています．フロッピィディスクの番地指定にはサーフェス番号，トラック番号，セクタ番号を使用します．サーフェス番号は，両面倍密度のフロッピィディスクを使用するときに必要です．「0」を指定すると表面，「1」を指定すると裏面になります．フロッピィディスクの最小読み書き単位はセクタで，3種類とも1セクタは256バイトからなります．

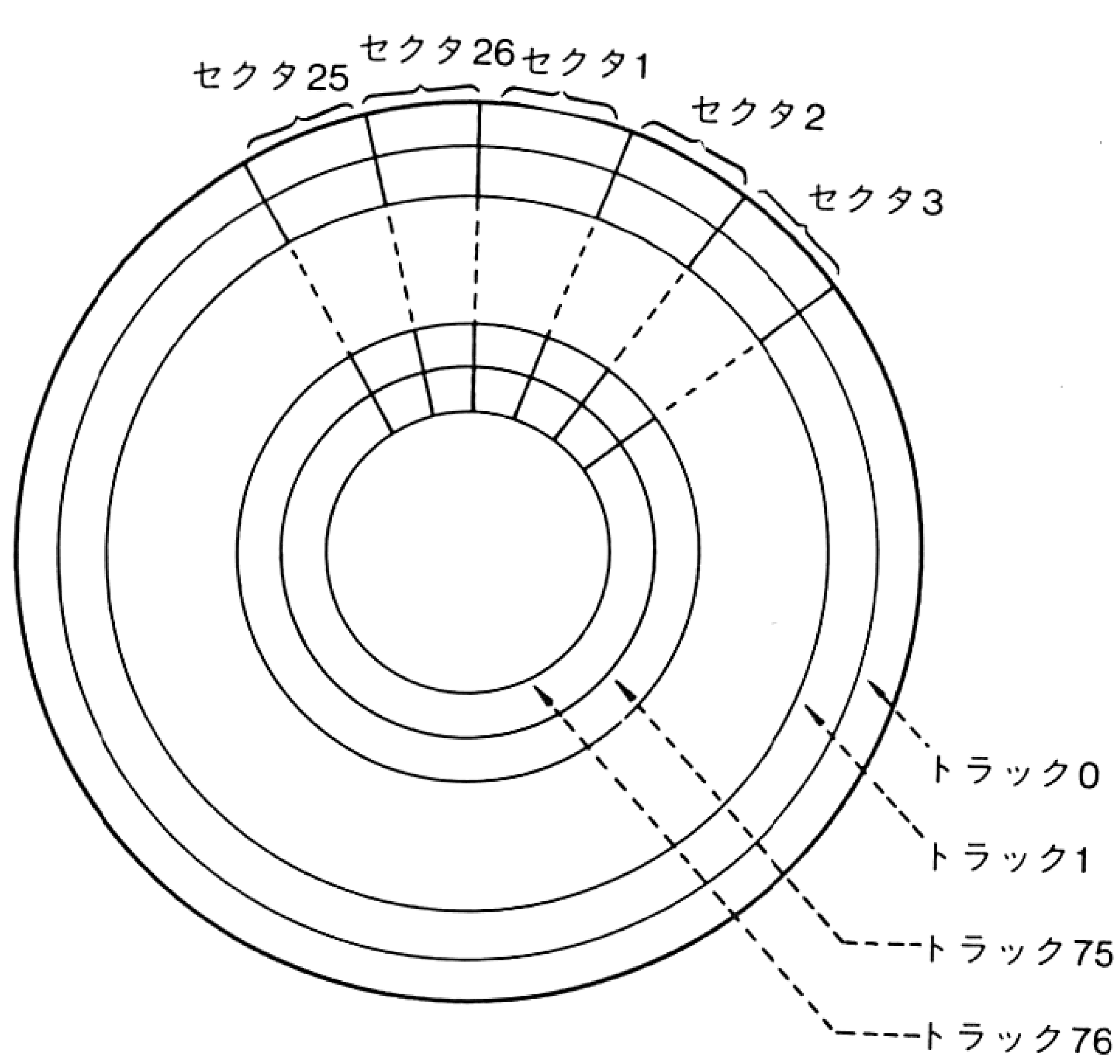
N$_{80SR}$-BASIC DISKバージョンで使用できる3種類のフロッピィディスクの構成は次のようになっています.



5.25インチ両面倍密度ミニフロッピィディスク



5.25インチ片面倍密度ミニフロッピィディスク



8インチ両面倍密度フロッピィディスク

図を見てもわかるように，トラック数，セクタ数はフロッピィディスクの種類によって異なります．これを表にまとめると次のようになります．

	サーフェス番号	トラック数	トラック番号	セクタ数	セクタ番号
5.25インチ両面倍密度	0か1	片面40(80)	0～39	16	1～16
5.25インチ片面倍密度	なし	35	0～34	16	1～16
8インチ両面倍密度	0か1	片面77(154)	0～76	26	1～26

資料5.2　トラックの割り当て

各トラックの割り当ては次のようになっています．

システムディスクとはDISKコードを含んだフロッピィディスクのことで，N_{80SR}-BASIC DISKバージョンを起動させる際に必要なものです．DISKコードとは，N_{80SR}-BASICインタプリタの拡張部分(フロッピィディスク関係の命令のサポート)のことです．それとは逆にDISKコードを含まないフロッピィディスクのことをデータディスクと呼びます．データディスクではN_{80SR}-BASIC DISKバージョンを起動させることはできません．

プログラムやデータは，システムディスクとデータディスクの両方に記憶させることができます．

(1)　5.25インチ両面倍密度ミニフロッピィディスク

システムディスク

サーフェス	トラック		セクタ		内容
	10進	16進	10進	16進	
0	0	0H	1	1H	IPL
	0	0H	2～16	2H～10H	DISKコード
	1	1H	すべて	すべて	DISKコード
	2～39	2H～27H	すべて	すべて	ユーザ領域
1	0～1	0H～1H	すべて	すべて	DISKコード
	2～17	2H～11H	すべて	すべて	ユーザ領域
	18	12H	1～12	1H～CH	ディレクトリ
	18	12H	13	DH	ID
	18	12H	14～16	EH～10H	FAT
	19～39	13H～27H	すべて	すべて	ユーザ領域

データディスク

サーフェス	トラック		セクタ	内　　　容
	10進	16進		
0	0～39	0H～27H	すべて	ユーザ領域
1	0～17	0H～11H	すべて	ユーザ領域
	18	12H	すべて	システムディスクと同じ
	19～39	13H～27H	すべて	ユーザ領域

(2) 5.25インチ片面倍密度ミニフロッピィディスク

システムディスク

トラック		セクタ		内　　　容
10進	16進	10進	16進	
0	0H	1	1H	IPL
0	0H	2～16	2H～10H	DISKコード
1～3	1H～3H	すべて	すべて	DISKコード
4～17	4H～11H	すべて	すべて	ユーザ領域
18	12H	1～12	1H～CH	ディレクトリ
18	12H	13	DH	ID
18	12H	14～16	EH～10H	FAT
19～34	13H～22H	すべて	すべて	ユーザ領域

データディスク

トラック		セクタ	内　　　容
10進	16進		
0～17	0H～11H	すべて	ユーザ領域
18	12H	すべて	システムディスクと同じ
19～34	13H～22H	すべて	ユーザ領域

(3)　8インチ両面倍密度フロッピィディスク

8インチ両面倍密度フロッピィディスクではサーフェス0，トラック0は1セクタ128バイトの単密度になっているため，BASICでは使用できません．

システムディスク

サーフェス	トラック		セクタ		内　　容
	10進	16進	10進	16進	
0	0	0H	すべて	すべて	未使用
	1	1H	1	1H	IPL
	1	1H	2～26	2H～1AH	DISKコード
	2	2H	すべて	すべて	DISKコード
	3～34	3H～22H	すべて	すべて	ユーザ領域
	35	23H	1～22	1H～16H	ディレクトリ
	35	23H	23	17H	ID
	35	23H	24～26	18H～1AH	FAT
	36～76	24H～4CH	すべて	すべて	ユーザ領域
1	0	0H	すべて	すべて	未使用
	1	1H	すべて	すべて	DISKコード
	2～76	2H～4CH	すべて	すべて	ユーザ領域

データディスク

サーフェス	トラック		セクタ	内　　容
	10進	16進		
0	0	0H	すべて	未使用
	1～34	1H～22H	すべて	ユーザ領域
	35	23H	すべて	システムディスクと同じ
	36～76	24H～4CH	すべて	ユーザ領域
1	0	0H	すべて	未使用
	1～76	1H～4CH	すべて	ユーザ領域

資料5.3　クラスタ

フロッピィディスクで読み書き可能な最小単位はセクタですが，N_{80SR}-BASICが管理する最小単位はクラスタと呼びます．

5.25インチミニフロッピィディスクは8セクタ，8インチフロッピィディスクは26セクタをひとまとめにして，1クラスタとします．クラスタとサーフェス，トラック，セクタの関係は

次の表のようになっています.

5.25インチ両面倍密度ミニフロッピィディスクの場合

クラスタ		サーフェス		トラック		セクタ	
10進	16進	10進	16進	10進	16進	10進	16進
0	0H	0	0H	0	0H	1～8	1H～8H
1	1H	0	0H	0	0H	9～16	9H～10H
2	2H	1	1H	0	0H	1～8	1H～8H
3	3H	1	1H	0	0H	9～16	9H～10H
4	4H	0	0H	1	1H	1～8	1H～8H
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
158	9EH	1	1H	39	27H	1～8	1H～8H
159	9FH	1	1H	39	27H	9～16	9H～10H

5.25インチ片面倍密度ミニフロッピィディスクの場合

クラスタ		トラック		セクタ	
10進	16進	10進	16進	10進	16進
0	0H	0	0H	1～8	1H～8H
1	1H	0	0H	9～16	9H～10H
2	2H	1	1H	1～8	1H～8H
3	3H	1	1H	9～16	9H～10H
4	4H	2	2H	1～8	1H～8H
⋮	⋮	⋮	⋮	⋮	⋮
68	44H	34	22H	1～8	1H～8H
69	45H	34	22H	9～16	9H～10H

8インチ両面倍密度フロッピィディスクの場合

クラスタ		サーフェス		トラック		セクタ	
10進	16進	10進	16進	10進	16進	10進	16進
0	0H	0	0H	0	0H	1～26	1H～1AH
1	1H	1	1H	0	0H	1～26	1H～1AH
2	2H	0	0H	1	1H	1～26	1H～1AH
3	3H	1	1H	1	1H	1～26	1H～1AH
4	4H	0	0H	2	2H	1～26	1H～1AH
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
152	98H	0	0H	76	4CH	1～26	1H～1AH
153	99H	1	1H	76	4CH	1～26	1H～1AH

資料5.4　ディレクトリ

フロッピィディスクに格納されているすべてのファイルの名前，属性，格納されている先頭の場所を表している表です．各ディレクトリは，16バイトからなります．各バイトは，次のような内容を持っています．

ディレクトリ

バイト		内　　容
10進	16進	
0〜5	0H〜5H	ファイル名
6〜8	6H〜8H	拡張子
9	9H	属性
10	AH	そのファイルが格納されている先頭のクラスタ番号
11〜15	BH〜FH	未使用

ファイル名の先頭バイト(0バイト目)の内容がFFH(16進数)のとき，そのディレクトリは未使用であることを示します．

00のとき，削除(KILL)されたファイルであることを示します．

属性は次のようになっています．

属性コード

値(16進数)	意　　味
0H	アスキー形式
1H	機械語ファイル
10H	アスキー形式，書き込み禁止
40H	アスキー形式，リードアフターライト
80H	非アスキー形式
90H	非アスキー形式，書き込み禁止
C0H	非アスキー形式，リードアフターライト

資料5.5　FAT

FATは，フロッピィディスクの記憶場所の使用状態を示す表です．FATは3セクタにわたって書き込まれており，各セクタには全く同じ内容が書かれています．フロッピィディスクをアクセスしたときに，FATの内容(3セクタ)がメモリにコピーされます．FATはフロッピィディスクの使用状態をクラスタ単位に管理していて，各クラスタに付き1バイトで，その使用状態を表します．

5.25インチ両面倍密度ミニフロッピィディスクの場合，各セクタの0～159バイトがクラスタ番号0～159にそれぞれ対応しています．すなわち，各バイトの持っている値が各クラスタの使用状態を示しています．

FAT

バイトのデータ(16進)	クラスタの使用状態
5.25インチ両面のとき0H～9FH 5.25インチ片面のとき0H～45H 8インチのとき 0H～99H	使用中．連続したクラスタの一部であり，後続するクラスタを持つ値が，後続するクラスタの番号を示している．
5.25インチのときC1H～C8H 8インチのとき C1H～DAH	使用中．連続したクラスタの最後のクラスタであり，下4(8インチのときは下5)ビットの内容が，そのクラスタで実際に使われているセクタの数を表す．
FEH	予約済みのクラスタで，ファイルとして使うことはできない．(DISKコード，IPL，ディレクトリ，FAT自身を含むクラスタがこれである．)
FFH	未使用

なお，表の説明で「連続したクラスタ」といいましたが，これは物理的に連続したクラスタを意味しません．N_{80SR}-BASIC DISKバージョンが管理するファイルは一般に物理的にとびとびのクラスタをつないで構成されているのです．

資料5.6　ID

IDには1セクタ(＝256バイト)が割り当てられています．各バイトは次のような意味を持っています．

属性	ファイル数	BASICテキスト
1バイト	1バイト	254バイト

IDの構造

- 最初の1バイトはそのフロッピィディスクの属性を示しています．値の意味は次のとおりです．

値(16進数)	意　味
0H	読み書き可
10H	書き込み禁止
40H	リードアフターライト

- 次の１バイトは起動時に設定するファイルの数を示しています．通常のシステムディスクでは〈ファイル数〉として255(16進数ではFFH)が指定されていて，このようなときは起動時に，

　　How many files (0-15) ?

　とたずねてきます．この〈ファイル数〉に０～15を指定するとN_{80SR}-BASIC DISKバージョンをスタートさせるときに，"How many files(0-15)?" を表示せずに，自動的にその値が設定されます．
- 次の254バイトは起動時に実行するBASICの命令を示しています．

IDは，〈属性〉が０，〈ファイル数〉が255(FFH)，〈BASICテキスト〉がすべて０か32(0Hか20H)が通常の状態です．

例)　　IDセクタ

00	05	72　75　6E　22　41　22　………　00

(r) (u) (n) (") (A) (")

この例のようにIDセクタの内容が書かれている場合，自動的に５個のファイルバッファが用意され，"A" という名前のプログラムが走り出します．

IDの〈属性〉はSET文によって書き換えることができます．

資料6　μPD8251の使い方

PC-8001MKⅡSRは，RS-232C通信ポート制御用として，μPD8251を使用しています．

μPD8251は，データ通信におけるシリアルインタフェースで，並列データを直列データに，また直列データを並列データに変換してデータの送受信を行います．

RS-232C 通信ポートを使用する際には，あらかじめこのμPD8251をモード設定しておきます．

INIT%文は，このモード設定を行うもので，書式は次のとおりです．

INIT%〈ポート番号〉,〈モード〉,〈コマンド〉

〈モード〉,〈コマンド〉は，それぞれ1バイトずつの整数値です．各ビットの意味を，図6.1～図6.2に示します．

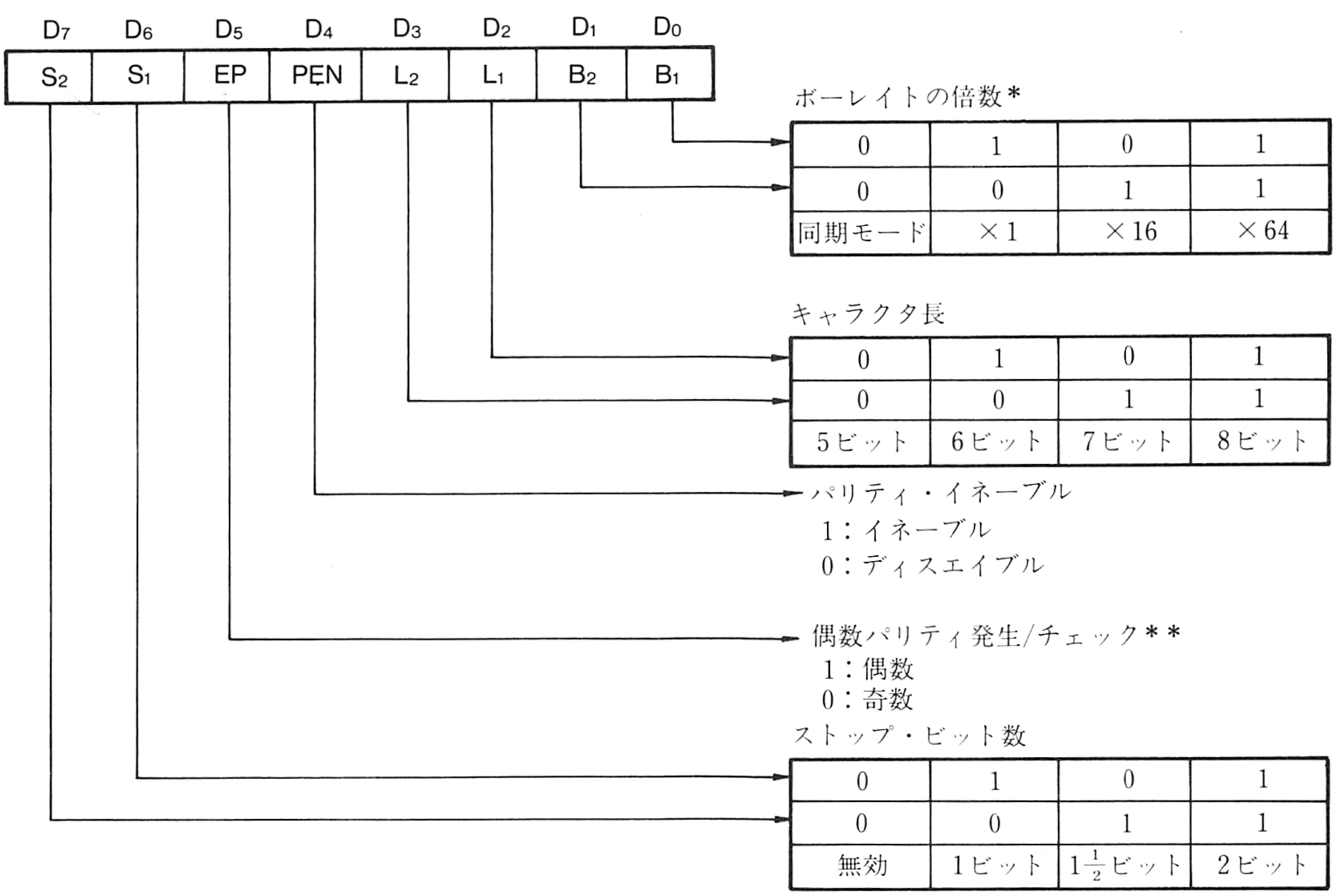


図6.1　〈モード〉

＊）　ボーレイトの倍数は通常"×16"または"×64"を使用します．

＊＊）　パリティがディスエイブル(D_4="0")のときは，"0"，"1"のどちらでもかまいません．

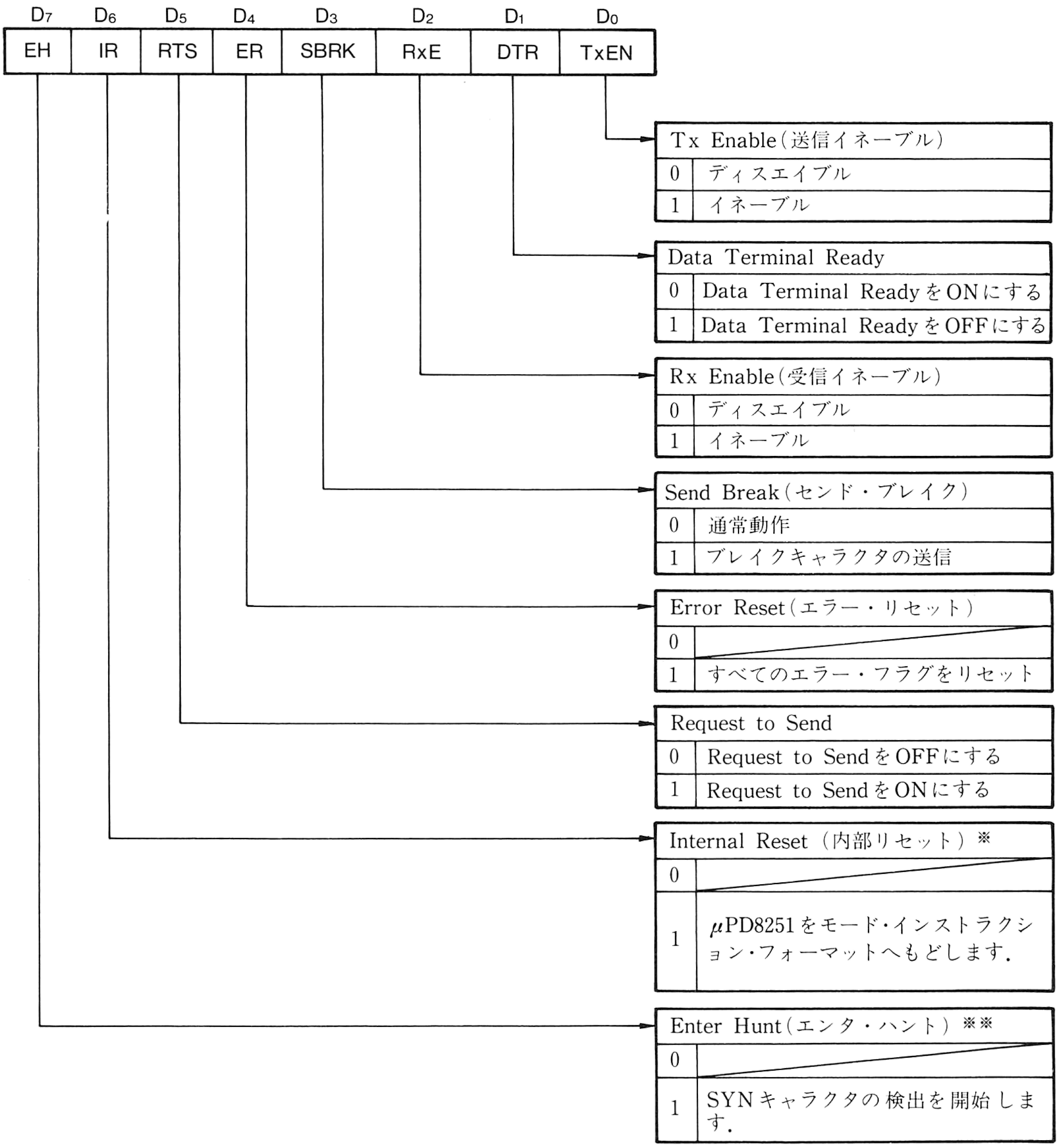


図6.2 〈コマンド〉

※　INIT%文ではInternal Resetは必ず0にしてください.

※※　非同期モードでは意味を持ちません.

注意：μPD8251には，同期モードと非同期モードがありますが，PC-8001MKⅡSRでは，非同期モードだけがサポートされています.

資料7　コントロールコード表

◦キー入力時の動作

16進	10進	キー入力	動　　　　作
02H	2	CTRL+B	カーソルを前の項目の先頭へ移動させます.
05H	5	CTRL+E	カーソルの位置から，その行の終わりまでを消します.
07H	7	CTRL+G	ブザーを鳴らします.
08H	8	CTRL+H	1文字を削除します. INS DELと同じ働きです.
09H	9	CTRL+I	タブ位置までカーソルを移動させます. タブ位置は8桁ごとに設定されています. TABキーと同じ働きです.
0AH	10	CTRL+J	行分割を行います.
0BH	11	CTRL+K	カーソルをホームポジション(左上スミ)に移動させます. SHIFT+HOME CLRと同じです.
0CH	12	CTRL+L	テキスト画面を消去します. HOME CLRと同じです.
0DH	13	CTRL+M	↵と同じです.
0EH	14	CTRL+N	カーソルを次の項目の先頭へ移動させます.
12H	18	CTRL+R	スペースを挿入します. SHIFT+INS DELと同じです.
1BH	27	ESC	実行を一時停止します.
1CH	28	⇆	カーソルを1つ右へ移動させます.
1DH	29	SHIFT + ⇆	カーソルを1つ左へ移動させます.
1EH	30	⇅	カーソルを上の行へ移動させます.
1FH	31	SHIFT + ⇅	カーソルを下の行へ移動させます.

◦画面表示のときの動作

PRINT文でCHR$関数を用いて出力します.

例)　　print chr$(7);

16進	10進	動　　　　作
07H	7	ブザーを鳴らします.
09H	9	タブ位置までカーソルを移動させます. タブ位置は8桁ごとに設定されています.
0AH	10	カーソルを1つ下の行へ移動させます.
0BH	11	カーソルをホームポジション(左上スミ)に移動させます.
0CH	12	テキスト画面を消去します.
0DH	13	カーソルを行の左端へ移動させます.
1CH	28	カーソルを1つ右へ移動させます.
1DH	29	カーソルを1つ左へ移動させます.
1EH	30	カーソルを上の行へ移動させます.
1FH	31	カーソルを下の行へ移動させます.

資料8　キャラクタコード

1　標準モード
（カタカナモード）

下位4ビット↓ ＼ 上位4ビット→	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P		p				ー	タ	ミ		
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	AK	SN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	HM	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	CL	→	,	<	L	¥	l	|			ャ	シ	フ	ワ	●	
D	CR	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	↓	/	?	O	_	o				ッ	ソ	マ	゜		

2　ひらがなモード

下位4ビット↓ ＼ 上位4ビット→	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P		p				ー	た	み		
1	SH	D1	!	1	A	Q	a	q			。	あ	ち	む		円
2	SX	D2	"	2	B	R	b	r			「	い	つ	め		年
3	EX	D3	#	3	C	S	c	s			」	う	て	も		月
4	ET	D4	$	4	D	T	d	t			、	え	と	や		日
5	EQ	NK	%	5	E	U	e	u			・	お	な	ゆ		時
6	AK	SN	&	6	F	V	f	v			を	か	に	よ		分
7	BL	EB	'	7	G	W	g	w			ぁ	き	ぬ	ら		秒
8	BS	CN	(	8	H	X	h	x			ぃ	く	ね	り	♠	
9	HT	EM	)	9	I	Y	i	y			ぅ	け	の	る	♥	
A	LF	SB	*	:	J	Z	j	z			ぇ	こ	は	れ	♦	
B	HM	EC	+	;	K	[	k	{			ぉ	さ	ひ	ろ	♣	
C	CL	→	,	<	L	¥	l	|			ゃ	し	ふ	わ	●	
D	CR	←	−	=	M	]	m	}			ゅ	す	へ	ん	○	
E	SO	↑	.	>	N	^	n	~			ょ	せ	ほ	゛		
F	SI	↓	/	?	O	_	o				っ	そ	ま	゜		

プリンタを標準モード，ひらがなモードに切り替える方法

テキスト画面の表示を標準モード(カタカナモード)あるいはひらがなモードに切り替えるには，CONSOLE文を使います．このCONSOLE文は，テキスト画面の表示モードの変更を行うだけでプリンタに対しては何の操作も行いません．したがって，LPRINT文やLLIST文，またCOPY文などでプリンタにひらがなを出力する場合は，プリンタをひらがなモードにする必要があります．

プリンタの標準モード(カタカナモード)，ひらがなモードを切り替えるには，次のようにして切り替えコマンドをプリンタに送ります．

- ひらがなモードにする

```
lprint chr$(27);"&";
```

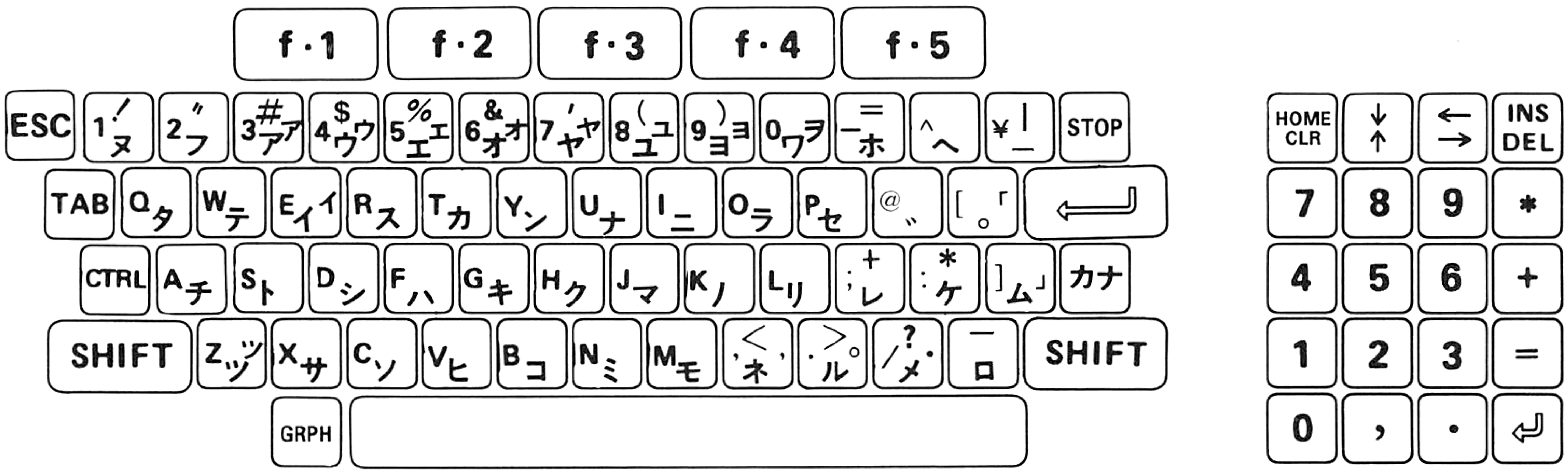
- 標準モードにする

```
lprint chr$(27);"$";
```

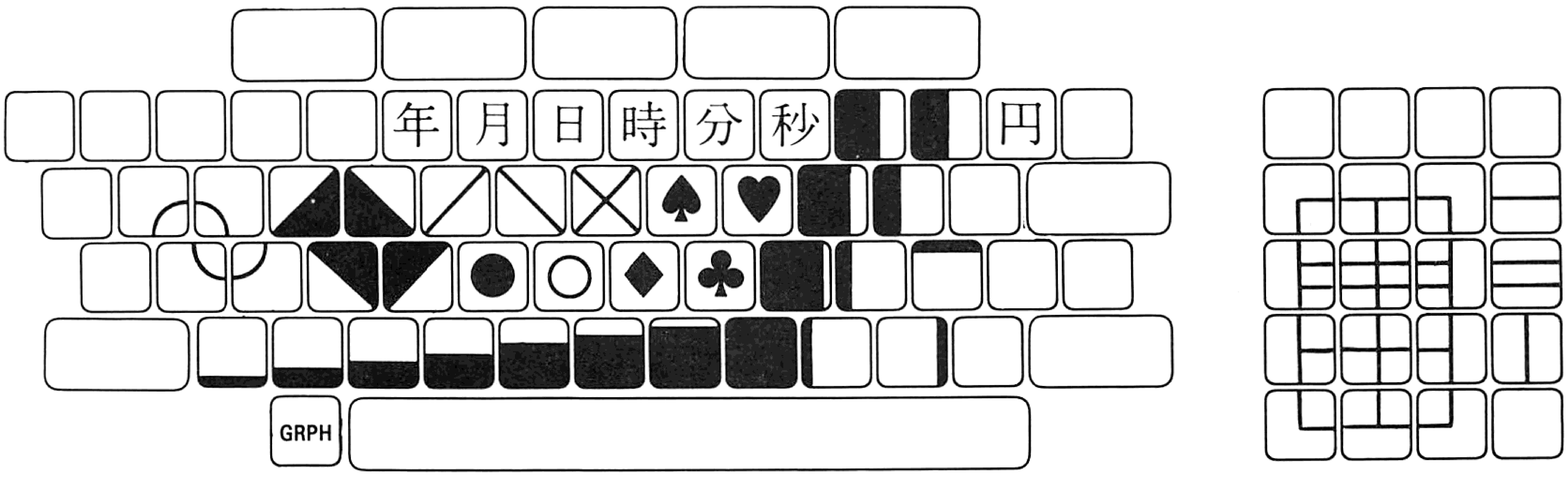
なお，この機能をサポートしているプリンタは次のとおりです．

PC-8023C, PC-8024, PC-8027
PC-8821, PC-8822, PC-8824, PC-8825
PC-PR101, PC-PR103, PC-PR201, PC-PR202
およびこれらの同等品

資料9　キーボードレイアウト



文字キーおよび特殊キー



グラフィックシンボルキー

資料10　エラーメッセージ

N_{80SR}-BASIC, N_{80}-BASICおよびN-BASICはプログラム実行時にプログラムの実行を中断させなければならないようなエラーを検出したとき，エラーメッセージを表示し，コマンドレベルに戻ります．

ダイレクトモードでのエラーメッセージの書式は，

XX

プログラムモードでの書式は，

XX in lllll

XXはエラーメッセージで，lllllはエラーが検出された行番号です．

ただし，以下のメッセージはエラーメッセージではありませんので，ON ERROR GOTO文によるエラー処理は行われません．

(1)　?Redo from start

プログラム中のINPUT文で，指定した変数の型がキーボードから入力した変数の型と一致していないことを表します．この場合は，もう一度正しく入力してください．

(2)　?Extra egnored

INPUT文で指定した変数の個数より，キーボードから入力されたデータの個数が多い場合，このメッセージが表示され，多い分は無視されます．

(3)　??

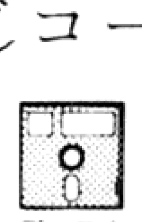
INPUT文で指定した変数の個数より，キーボードから入力されたデータの個数が少ない場合，このメッセージが表示され足りないデータの入力を促します．

(4)　1 copies of allocation bad on drive〈ドライブ番号〉，または，

2 copies of allocation bad on drive〈ドライブ番号〉

このメッセージは，フロッピィディスクを使用している場合，そのフロッピィディスクの内容が壊れかかっているときに表示されます．このような状態になったときは，そのフロッピィディスクのコピーをとることをお薦めします．

次に，N_{80SR}-BASIC, N_{80}-BASIC, N-BASICで表示されるエラーメッセージをアルファベット順に表記し，エラーメッセージの意味，エラーが起こった原因とその解説をします．

コード番号はエラー別に決まっていて，そのエラーが起こったとき，ERR関数に代入されます．上記3つのBASICとも同じコード番号を持ちます．また，DISKバージョン使用時だけに出力されるエラーメッセージは 

マークで表しています．

Bad allocation table　　コード番号64 ディスク

意味　フロッピィディスク上のFAT(フロッピィディスクの中のファイル管理をしている部分)が壊れている.

原因
- フロッピィディスクが入っていないドライブに対して，ロードやセーブを行った.（ミニフロッピィディスクユニット使用時のみ）
- フロッピィディスクのふたがしまっていない.（ミニフロッピィディスクユニット使用時のみ）
- フロッピィディスクに対しOPEN文を実行しファイルを書き込んだあと，CLOSE文を実行しなかった.
- DSKO$文や機械語モニタ(MON)の **CTRL** ＋ **W** コマンドでFATを書き直してしまった.

解説　ミニフロッピィディスクユニットを使用しているとき，フロッピィディスクを入れずにあるいはドライブのふたを閉めずにフロッピィディスクをアクセスすると，ドライブのアクセス表示用LEDランプがつきっぱなしになり，しばらくすると上記のエラーメッセージが表示されます.

また，OPEN文を実行したあとにCLOSE文を実行しなかったり，DSKO$文や機械語モニタでFATを書き直してしまった場合，そのフロッピィディスクに対してファイルの入出力は不可能になります.

Bad drive number　　コード番号65 ディスク

意味　フロッピィディスクのドライブ番号の指定が誤っている.

原因
- システムに接続されていないドライブ番号を指定した. たとえば，フロッピィディスクドライブが2ドライブだけのシステムに対し，ドライブ番号に3を指定した.

解説　システムの起動時に，指定可能なドライブ番号が決定されます.（起動時に電源が入っていなかったフロッピィディスクドライブは，接続されていないものとみなされます.）それよりも大きい番号を指定したときにこのメッセージが表示されます.

Bad File Data　　コード番号25

意味　カセットテープ上にあるデータの形式が間違っている.

原因
- カセットテープに書かれたデータの型がINPUT #文で読もうとしている変数の型と一致していない.

解説　カセットテープからデータを読み込む場合，データの型は書き込んだときのデータの型と一致していなければなりません.

Bad file name　　コード番号62 ディスク

意味　ファイル名の指定の仕方が誤っている.

原因　• LOAD, SAVE, KILL, NAME, OPEN, RUNなどの命令でファイル名を指定するとき，ファイル名に使えない文字や記号がある．

解説　ファイル名の中にコントロールコード(キャラクタコード0～31)を含むことはできません．また，ファイル名の先頭がキャラクタコード255の文字であってはなりません．

Bad file number　　コード番号52

意味　使用できないファイル番号を使おうとした．

原因　•起動時に指定したファイル番号より上のファイル番号をOPEN文，PRINT#文，INPUT#文，GET#文，PUT#文等で使っている．

解説　ファイル番号は，「How many files(0-15)?」が表示された時に入力したファイル数しか使用できません．指定できるファイル番号の範囲は1から15までです．

Bad file mode　　コード番号54

意味　シーケンシャルでオープンしたファイルに対してランダムアクセスをしようとした．またはその逆を行おうとした．(PC-8001MKⅡSRではこのエラーは発生しません．)

Bad surface/track/sector　　コード番号66

意味　サーフェス，トラック，セクタ番号の指定が誤っている．

原因　• DSKI$関数，DSKO$文のパラメータ中のサーフェス番号，トラック番号，セクタ番号が指定できる値の範囲にない．

解説　DSKI$関数やDSKO$文で直接フロッピィディスクに対して入出力を行うとき，フロッピィディスクドライブやシステムディスクの種類によって，指定できるパラメータの範囲は異なります．この値はDSKF関数を使って調べることができます．

Can't continue　　コード番号17

意味　CONTコマンドでプログラムの続行ができない．

原因　•プログラムをストップさせてから編集をすると，CONTコマンドによるプログラムの再開・続行ができない．

　　　•プログラム中にステートメントとしてCONTコマンドが書いてある．

解説　プログラムをストップさせて変数の値を調べたり，変数に値を代入したり，LISTコマンドを実行してもCONTコマンドによる実行の再開は可能ですが，プログラムを一部でも書き換えたり，CLEAR文を実行したり，画面に表示したプログラム上にカーソルを移動して[↵]を押しただけでもこのエラーの原因となります．

Communications Buffer Overflow　　コード番号27

意味　周辺機器との入出力のためのバッファがオーバーフローした．

原因　• RS-232C通信ポートの入出力バッファ(127バイト)のデータが容量を超えた．

・アスキー形式のプログラムファイルをロードしようとしたときに，入力バッファがオーバーフローした．

解説　RS-232C通信ポートの入出力バッファは127バイト(127文字)からなり，データが127文字を超えると上記のエラーメッセージが表示されます．PORT関数を使ってバッファ内の文字数を調べてバッファがある程度いっぱいになったら，INPUT%文などでデータの読み出しを行ってください．アスキー形式のプログラム中に，1行の長さがかなり長いものがあった場合，それをロードする際にこのエラーが出ることがあります．

Deleted record　　コード番号67　ディスク

意味　消去済みのレコードを読み書きしようとした．(PC-8001MKⅡSRでは，このエラーは発生しません．)

Direct statement in file　　コード番号63　ディスク

意味　アスキー形式のBASICプログラムファイル中に行番号のない行が存在した．

原因　・SAVEコマンドでアスキーセーブしたBASICプログラム以外のアスキー形式ファイルをロードした．

・SAVEコマンドでアスキーセーブしたBASICプログラムファイルの内容が何らかの原因で破壊されているのに，そのファイルをロードしようとした．

解説　BASICプログラムの各行は必ず行番号で始まりますが，アスキー形式のファイルをフロッピィディスクからロードするときには，どんなデータが読み込まれるか予測できません．したがって，BASICでは各行の先頭に行番号が正しく付けられているかをチェックし，行番号がなければこのエラーメッセージを表示します．

Disk already mounted　　コード番号59　ディスク

意味　MOUNTされているフロッピィディスクに対してMOUNTコマンドを行った．

原因　・MOUNTされているフロッピィディスクに対してMOUNTコマンドを行った．

解説　N-BASIC DISKバージョンでは，既にMOUNTされているフロッピィディスクに対して，MOUNTコマンドを実行することはできません．(N_{80SR}-BASIC, N_{80}-BASICではこのエラーは発生しません．)

Disk BASIC Feature　　コード番号26

意味　使用できない命令を実行した．

原因　・DISKバージョンが起動していないときにDISKバージョンの命令を実行した．

・TALK文などの拡張用の命令を実行した．

解説　DISKバージョン用の命令を実行する場合は，DISKバージョンを起動させてから実行

してください．

Disk full　　コード番号60　ディスク

意味　フロッピィディスク上に書き込むスペースがないのに書き込む命令をした．

原因　•フロッピィディスク上の空き領域よりも大きなプログラムをセーブしようとした．

　　　•ファイルを書き込むときに，フロッピィディスクがいっぱいになった．

解説　このエラーが出たときには，そのフロッピィディスクにはそれ以上のデータは書き込めませんので，ファイルをオープンしているときはCLOSE文を実行したあとに他のフロッピィディスクに取り替えてください．N-BASIC DISKバージョンを使用している場合は，REMOVEコマンドを実行してから他のフロッピィディスクに取り替えてください．フロッピィディスク上の空き領域の大きさは，DSKF関数で確認できます．

Disk not mounted　　コード番号56　ディスク

意味　MOUNTされていないフロッピィディスクを読み書きした．

原因　•MOUNTされていないフロッピィディスクに対して読み書きを行った．

解説　N-BASIC DISKバージョンでフロッピィディスクを読み書きする場合には，最初に必ずMOUNTコマンドを実行してから読み書きするようにしてください．

Disk I/O error　　コード番号57　ディスク

意味　フロッピィディスクとの入出力中にエラーが発生した．

原因　•フロッピィディスクが初期化されていない．

　　　•フロッピィディスクに傷や汚れがある．

　　　•フロッピィディスクユニットに故障がある．

　　　•ライトプロテクトされているフロッピィディスクに書き込みを行った．

解説　このエラーがある特定のフロッピィディスクに関してのみ出る場合には，フロッピィディスクに傷や汚れが付いていると考えられます．また，フロッピィディスクに関係なくこのエラーが出る場合には，フロッピィディスクユニットが故障している可能性があります．

注意：コピープロテクトがかけられたソフトをコピーしようとするとこのエラーが出ることがありますが，これは故障ではありません．

Disk offline　　コード番号73　ディスク

意味　フロッピィディスクに対して入出力ができない．

原因　•フロッピィディスクが入っていないドライブに対し，ロードやセーブを行った．

　　　•フロッピィディスクドライブのふたがしまっていない．

解説　このエラーは8インチフロッピィディスクユニットを使用するときのみ出ます．

Division by zero　　コード番号11

意味　0による除算が実行されて，その結果がオーバーフローした．

原因　• 未定義の変数(初期値として0になっている)で除算を行った．

• 演算の結果除数となる変数が0になっている．

• TAN関数の引数がπ/2になっている．

• 0に対して負のべき乗を行った．

解説　エラーが出たときに，除数に使っている変数をPRINT文を実行して値を確かめます．0であればなぜ0になるか，プログラムの中でその変数を使って演算を行っている部分を検討してみます．除算には実数除算"/"，整数除算"¥"，剰余"MOD"の3つがあります．

FIELD overflow　　コード番号50

意味　FIELD文において256バイトを超える大きさの領域を指定した．

原因　• FIELD文において1つの文字変数のフィールド幅，もしくは1つのバッファのフィールド幅の合計値が256文字を超えている．

解説　1つのバッファの合計文字数は256文字までです．FIELD文を使用する場合は，1つのバッファに対するフィールド幅の合計が256文字を超えないように設定してください．

File already exists　　コード番号58

意味　NAME文によって変更しようとした新ファイル名が既に存在している．

原因　• NAME文によって指定した新ファイル名が既にフロッピィディスクに存在している．

解説　NAME文によって指定した新ファイル名がフロッピィディスクに存在しているとこのエラーが発生します．他のファイル名を使用するか，フロッピィディスクに存在するファイルをKILLコマンドで削除するなどしてから実行してください．

File already open　　コード番号55

意味　同じファイル番号で2度OPEN文を実行しようとした．または，既にオープンされているファイルに対してOPEN文，KILLコマンドなどを実行しようとした．

原因　• OPEN文の実行後CLOSE文を実行するのを忘れている．

• KILLコマンド実行の前にCLOSE文を実行するのを忘れている．

解説　オープン中のファイル番号を，他のファイルをオープンするために使うことはできません．またオープンされているファイルをクローズせずに再びオープンすることもできません．いずれの場合も，ファイルをオープンしたらクローズするのを忘れないようにします．KILLコマンドを実行するときも同様です．

File not found　　コード番号53

意味　指定したファイルが見つからない．

原因　• LOADコマンド，SAVEコマンド，KILLコマンド，OPEN文などで指定したファイル名が指定したフロッピィディスク上にない．

解説　このようなエラーをなるべく避けるために，ファイル名はわかり易い名前を付け，拡張子を付けておくとよいでしょう．

File not OPEN　　コード番号71

意味　オープンされていないファイルを参照しようとした．

原因　• まだOPEN文でファイルを割り当てていないファイル番号を使ってPRINT# 文などを実行した．

解説　PRINT#文，INPUT#文，PRINT# USING文，LINE INPUT#文，INPUT$関数，GET#文，PUT#文で使うファイル番号は，事前にOPEN文によって対応するファイルを割り当てておかなければなりません．

File write protected　　コード番号72

意味　書き込み禁止属性が付けられているファイルに書き込もうとした．

原因　• 書き込みを行おうとしたフロッピィディスクやファイルに書き込み禁止属性が付けられている．

• ミニフロッピィディスクにライトプロテクトシールが貼ってある．

• 8インチフロッピィディスクにライトプロテクトノッチが切り込んである．

解説　書き込もうとしたフロッピィディスクまたはファイルにSET文で書き込み禁止属性が付けられている場合，書き込み禁止属性を解除します．また．ミニフロッピィディスクにライトプロテクトシールが貼ってある場合はシールをはがし，8インチフロッピィディスクのライトプロテクトノッチに切り込みが入っている場合は，ライトプロテクトノッチに銀色のシールを貼ってください．

Illegal direct　　コード番号12

意味　ダイレクトモードで使用できない文を実行しようとした．

原因　• コマンドとして使えないステートメントをダイレクトモードで使った．

解説　このエラーが表示されたステートメントはプログラムモードでしか使用できません．（たとえば，DEF FN文をダイレクトモードで使用した場合）．

Illegal function call　　コード番号 5

意味　ステートメントや関数において機能の呼び方が間違っている．

原因　• パラメータの値がおかしい．

・PRINT USING文での数値の桁指定が24桁を超えている．

・RENUMコマンドで行の順番を入れ替えようとした．

解説　原因が多岐にわたり，頻発するエラーの１つです．各関数やステートメントの使い方を区別して覚えることが大切です．プログラム中のエラーが生じるのはほとんどの場合，パラメータに使用している変数が演算の結果予定外の値になってしまうことによります．これはエラーが出た直後にその変数の値を表示してみればすぐわかります．パラメータに使う変数名は極力わかり易いものにして，他と区別するようにすると良いでしょう．

Input past end　　コード番号61

意味　ファイル中のすべてのデータを読み尽くした後にINPUT#文，GET#文等が実行された．

原因　・INPUT#文，INPUT$関数，LINE INPUT#文，GET#文を実行する回数がファイルのデータ数より多い．

解説　INPUT#文，GET#文などでデータを次々と読み出すとき．ファイル中のデータ数以上読ませないようにします．ファイル中のデータ数がわからないときは，EOF関数あるいはLOF関数を使ってデータの最後を確かめてからINPUT#文，GET#文を実行します．

Internal error　　コード番号51

意味　BASIC内部にエラーが生じた．

原因　・BASICプログラムに他のエラーメッセージで表せないエラーが発生した．

解説　ハードウェアの故障(たとえば，シンセサイザーICの故障等)があるときにこのエラーが発生する場合があります．

Line buffer overflow　　コード番号23

意味　プログラムの入力バッファがオーバーフローした．

原因　・プログラム中の１行を入力したときに，変換された内部コードが318文字を超えた．

解説　入力されたプログラムはメモリ内に記憶する前に内部コードに変換されます．このエラーは変換されたコードが変換バッファの長さを超えた場合に表示されます．このようなときは，行を分割してください．

Missing operand　　コード番号22

意味　ステートメント中必要なパラメータが指定されていない．

原因　・コマンドや関数で省略できないパラメータや引数が欠けている．

・代入文の右辺がない．

• 演算子があるのに被演算子がない.

解説　コマンド・ステートメント，関数の使い方や第1章の式と演算などをもう一度確認してください.

NEXT without FOR　　コード番号1

意味　FOR～NEXT文が正しく対応していない(NEXTが多い).

原因　• FORの数がNEXTの数と合っていない.

• FOR～NEXTループの中にGOTO文やGOSUB文で飛び込んでくる.

• FOR～NEXT文の多重ループが正しい入れ子になっていない.

解説　FORの数とNEXTの数は必ず一致するようにプログラミングをします．したがって，IF文中にNEXT文を置くことができない場合があります．入れ子構造を理解するためには，NEXT文の後に制御変数(カウンタ)を付けておき，複数をまとめて指定しない方が見やすくなります.

No RESUME　　コード番号19

意味　RESUME文がない.

原因　• エラー処理ルーチンの中にRESUME文がない.

解説　エラー処理ルーチンは必ずRESUME文，END文，ON ERROR GOTO 0のどれかで終わっていなければなりません.

Out of DATA　　コード番号4

意味　読むべきデータがないのにREAD文が実行された.

原因　• DATA文のデータ数が足りない.

• RESTORE文の使い方が間違っている.

解説　同じデータを何度も読ませる場合や，データをグループごとに違った行にまとめておいて扱うときはRESTORE文を使いますが，RESTORE文で指定した行番号が間違っているとこのようなエラーが発生します．READ文とDATA文の対応を正しく行ってください.

Out of memory　　コード番号7

意味　メモリ容量が足りなくなった.

原因　• プログラムが長すぎてメモリの中に収まりきらない.

• プログラムは収まっても，それを走らせるのに必要なメモリ(変数，スタック用など)が足りない.

• 配列が大きすぎて，メモリ上にその領域をとることができない.

• FOR文やGOSUB文によるネスティング(入れ子)が深くなりすぎて，スタックがいっぱいになった.

・機械語用のメモリ領域を(既にプログラムやファイルで使っている領域を壊す程)大きくとろうとした.

・PAINT文で複雑な図形をペイントしようとした.

解説　このエラーが表示されたら，FRE関数を使ってフリーメモリを調べてください．プログラムが短いのにフリーメモリが異常に少ない場合は，起動時に多くのファイルバッファをとっていたり，機械語領域を指定したまま忘れている可能性があります．また変数や配列はできるだけ整数型にします.

また，間違って10 GOSUB 10のように無限にサブルーチンコールを繰り返してしまうと，戻り番地を覚えるためのスタックが無限にふくれ上り，メモリが足りなくなってしまいます.

Out of string space　　コード番号14

意味　文字列を格納するメモリ領域がない.

原因　・文字列を扱っているときにメモリ領域がなくなった.

解説　ストリング領域の初期設定は N_{80SR}-BASICでは512，N_{80}-BASICおよびN-BASICでは300になっています．多くの文字列演算を行ったり，大きな文字配列を用いる場合は，CLEAR文でこの領域を大きく指定しないとエラーが起こります.

Overflow　　コード番号 6

意味　入力された数値，代入される数値，演算結果などが許される範囲を超えている.

原因　・整数演算の結果や整数型変数に代入する値が－32768から32767の範囲外である.

・実数演算などの結果が－1.70141E＋38から1.70141E＋38の範囲にない.

・PEEK関数，POKE文，OUT文，DIM文などで指定するパラメータや添字の値が正しい範囲にない.

解説　変数の型とその範囲，コマンド・ステートメントのパラメータの範囲，関数で扱える範囲，論理演算の範囲などを把握しておくことが必要です.

Port not initialized　　コード番号28

意味　インタフェース用のLSIの機能設定がされていない.

原因　・イニシャライズを行っていない状態でRS-232Cインタフェースを使用した.

解説　RS-232C等のインタフェースを使用する場合，INIT%文を使用してインタフェース用のLSIに機能設定を行ってください.

Position not on screen　　コード番号24

意味　指定したカーソル位置などが，画面の範囲外になっている．(PC-8001MKⅡSRでは，このエラーは発生しません.)

Redimensioned array　　　コード番号10

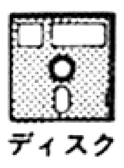
意味　配列を二重定義しようとした．

原因　• 一度宣言した配列を再定義した．

　　　• 宣言をしないで配列を使っても，添字の上限を10として自動的に宣言したことになるので再定義はできない．

解説　一度宣言した配列は，NEW コマンド，RUN コマンド，CLEAR 文などの命令が実行されるまでは，再定義できません．これらの命令が実行されると他の変数もすべてクリアされてしまいます．特に配列の添字の上限だけを変えることはできませんので，こういう場合は，別の名前の配列を新しく宣言して定義するしかありません．

　　　特定の配列だけ再定義したいときは，ERASE 文を使用してその配列を消去してから DIM 文を実行してください．

Rename across disks　　　コード番号68 

意味　NAME 文において，異なるフロッピィディスクドライブ間でファイルの名前を変えようとした．

原因　• NAME 文で指定するドライブ番号が〈旧ファイル名〉と〈新ファイル名〉とで異なっている．

解説　ドライブ番号は〈旧ファイル名〉と〈新ファイル名〉の両方に指定しなければなりません．

　　　ドライブ番号を省略した場合，ドライブ 1 が指定されたとみなされます．

RESUME without error　　　コード番号20

意味　エラー処理ルーチンに制御を移さなかったのに RESUME 文がある．

原因　• GOTO 文や GOSUB 文でエラー処理ルーチンの中へ分岐している．

　　　• メインルーチン終了後に END 文がなく，後に続いているエラー処理ルーチンの実行が始まってしまう．

解説　メインルーチンの終わりには END 文をおき，エラー処理ルーチンに不要意に飛び込むのを避けます．

RETURN without GOSUB　　　コード番号 3

意味　GOSUB～RETURN 文が正しく対応していない．(RETURN が多い)

原因　• サブルーチンへ GOTO 文で飛び込んでいる．

　　　• メインルーチン終了後に END 文がなく，後に続いているサブルーチンの実行が始まってしまう．

解説　メインルーチンの終わりには END 文をおくようにします．また，サブルーチンとサブルーチンの間には必ず RETURN 文をおくようにしてください．

Sequential after PUT　　コード番号69

意味　PUT#文のあとにシーケンシャル出力を行った．

原因　• ランダムファイルに対してPUT#文でデータを書き込んだあと，PRINT#文を実行した．

解説　ファイルにデータを書き込む場合，PRINT#文とPUT#文を混在して使用することはできません．シーケンシャルファイルとランダムファイルは区別して使用してください．

Sequential I/O only　　コード番号70

意味　シーケンシャルファイルの入出力以外は行ってはならない．

原因　• MERGEコマンドなどアスキー形式のファイルしか指定できない命令においてバイナリ形式のファイルを指定した．

解説　MERGE文はメモリ上のプログラムと〈ファイル名〉によって指定されたプログラムファイルを1つにしてメモリ上に置きますが，指定するファイルは必ずアスキーセーブされていなければなりません．

String formula too complex　　コード番号16

意味　文字式が複雑すぎる．

原因　• カッコのネスティングが深すぎるなど，文字式があまりに複雑で解析できない．

解説　めったに起こるエラーではありませんが，複雑すぎる文字式は2つ以上に分割してください．

String too long　　コード番号15

意味　文字列が長すぎる．

原因　• 文字演算の結果，文字列の長さが255を超えてしまった．

解説　文字列の長さは255までと決まっているので，A$＝A$＋"LONG"などの演算を繰り返しているとA$の長さが256になった時点でエラーが発生します．文字列演算が無限ループの中にあったり，演算の要素になる文字列(特に文字型変数)が予定外の長さになっているということのないように気をつけます．どうしても256以上の長さのストリングを扱いたければ，2つ以上の変数名を使って分割処理しなければなりません．

Subscript out of range　　コード番号 9

意味　配列の添字が，DIM文によって宣言されたときの大きさの範囲を超えている．

原因　• 添字に変数を使っている場合，この変数の値が演算の結果大きくなり過ぎている．
　　　• 配列の次元数を誤っている．

解説　エラーが起こったらすぐに，その添字として使っている変数の内容を表示して調べます．予定外に大きければ，その変数を使った演算が原因です．

Syntax error　　コード番号2

意味　文の記述が文法どおりになっていない．

原因
- タイプミスでBASIC文法に合わないものがプログラムにまぎれ込んでいる．
- 関数や数式が代入文の左辺にあったり，ステートメントのように単独で使われている．
- 変数名が英字で始まっていない，あるいは変数名に予約語を使用している．
- マルチステートメントの区切り記号":"が抜けている．
- 行番号が0から65529の範囲にない．
- 行番号の指定に変数を使おうとしている．
- IF文中で，対応するTHENのないELSEが使われている．
- コマンドのパラメータや関数の引数の個数に過不足がある．
- プログラム編集中に2つの行がつながってしまっている．

解説　エラーがでたときにLIST.を実行すると，ほとんどの場合エラーのある行を示してくれるもので，この中で原因となる間違いを探します．単純なタイプミスなどはすぐわかりますが，次のようなものはなかなか気がつかないことがあります．
- 1とI，0とO，ピリオド"."とコンマ","，コロン":"とセミコロン";"などの違い．
- 複雑な数式でカッコが正しく対応していない．
- 2つの行が接合している．

Tape read ERROR　　コード番号29

意味　カセットテープからの読み込み時にエラーが発生した．

原因
- テープレコーダの不良，テープの不良，録音時の不良，ロードレベルの不良等の原因で読み込んでいるデータのフォーマットに誤りがあった．

解説　このエラーが発生した場合には次の点を再確認してください．
- テープレコーダの音量調節は適当か(テープレコーダの音量，音質ツマミを調節してみる)．
- テープレコーダとの相性はよいか(他のテープレコーダを使用して読み込んでみる)．
- テープへの録音状態が悪くないか(他のCPU本体を使用して読み込んでみる)．

Type mismatch　　コード番号13

意味　式の左辺・右辺，関数などの引数などにおいて変数の型が一致していない．

原因　・文字型変数に数値を代入しようとした．
・数値変数に文字列を代入しようとした．
・FOR文の制御変数に倍精度実数型を使おうとした．

解説　CHR$関数とASC関数，STR$関数とVAL関数のような関数の使い方を確認してください．変数や定数について正しく理解しておくと間違いを見つけるのが容易になります．
また，FOR文の制御変数は整数型と単精度実数型しか使えません．

Undefined line number　　コード番号8

意味　飛び先として必要とされる行番号が存在しない．

原因　・RENUMコマンド実行時に参照すべき行番号がない．
・GOTO文，GOSUB文などの分岐先の行番号が存在しない．
・RESTORE文，RUNコマンドで指定した行番号が存在しない．
・存在しない行を消去しようとした．

解説　GOTO文の分岐先の行をプログラム編集時に消去したまま忘れていることがあります．編集で一行消してしまうときは，そこに分岐する命令がプログラム中にないことを確かめてください．

Undefined user function　　コード番号18

意味　DEF FN文によって定義されていないユーザ定義関数を引用しようとした．

原因　・DEF FN文で関数を定義せずにユーザ定義関数を呼んだ．

解説　ユーザ定義関数を呼ぶためには，その前にDEF FN文で関数を定義する必要があります．定義の方法は2章のDEF FNを参照してください．

Unprintable error　　コード番号21

意味　メッセージの定義されていないエラーを出そうとした．

原因　・ERROR文により，定義されていないエラーコード番号のエラーを発生した．

解説　"Unprintable error"は，BASICの拡張やユーザ定義のために残されたコード番号に割り当ててあるもので21の他30～49及び74～255のコード番号に対応しています．
ERROR文でユーザが定義したエラーメッセージを出すには，エラー処理ルーチンでその処理を行わなければなりません．

資料11　漢字コード表

漢字は16×16ドットで表示されますので，字体の一部が省略されているものもあります．

コードは16進数で表現されています．たとえば，“亜”のコードは3020H＋1＝3021Hとなります．空白文字のコードは2121Hです．

注意： 漢字コード××20H，××7FHおよびこの表に表示されていないコードの漢字は定義されていません．

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
半角文字	0020		!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
	0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	—
	0060		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0070	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	
	0080		。	「	」	、	・	を	ぁ	ぃ	ぅ	ぇ	ぉ	ゃ	ゅ	ょ	っ
	0090	ー	あ	い	う	え	お	か	き	く	け	こ	さ	し	す	せ	そ
	00A0		。	「	」		・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
	00B0	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
	00C0	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
	00D0	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
	00E0	た	ち	つ	て	と	な	に	ぬ	ね	の	は	ひ	ふ	へ	ほ	ま
	00F0	み	む	め	も	や	ゆ	よ	ら	り	る	れ	ろ	わ	ん	゛	゜
1/4角文字	0100		SH	SX	EX	ET	EQ	AK	BL	BS	HT	LF	HM	CL	CR	SO	SI
	0110	DE	D1	D2	D3	D4	NK	SN	EB	CN	EM	SB	EC	→	←	↑	↓
	0120		!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
	0130	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0140	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0150	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	—
	0160		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0170	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	
	0180	▁	▂	▃	▄	▅	▆	▇	█	▏	▎	▍	▌	▋	▊	▉	┼
	0190	┴	┬	┤	├	▔	─	│	▕	┌	┐	└	┘	╭	╮	╰	╯
	01A0		。	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
	01B0	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
	01C0	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
	01D0	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
	01E0	═	╞	╪	╡	◢	◣	◥	◤	♠	♥	♦	♣	●	○	╱	╲
	01F0	×	円	年	月	日	時	分	秒								
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
記号	2120			、	。	，	．	・	：	；	？	！	゛	゜	´	｀	¨
	2130	＾	￣	＿	ヽ	ヾ	ゝ	ゞ	〃	仝	々	〆	〇	ー	―	‐	／
	2140	＼	〜	‖	｜	…	‥	‘	’	“	”	（	）	〔	〕	［	］
	2150	｛	｝	〈	〉	《	》	「	」	『	』	【	】	＋	−	±	×
	2160	÷	＝	≠	＜	＞	≦	≧	∞	∴	♂	♀	°	′	″	℃	￥
	2170	＄	¢	£	％	＃	＆	＊	＠	§	☆	★	○	●	◎	◇	
	2220		◆	□	■	△	▲	▽	▼	※	〒	→	←	↑	↓	〓	
英・数字	2330	0	1	2	3	4	5	6	7	8	9						
	2340		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	2350	P	Q	R	S	T	U	V	W	X	Y	Z					
	2360		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	2370	p	q	r	s	t	u	v	w	x	y	z					
ひらがな	2420		ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
	2430	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
	2440	だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
	2450	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
	2460	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
	2470	ゐ	ゑ	を	ん												
カタカナ	2520		ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
	2530	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
	2540	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ
	2550	バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ
	2560	ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ
	2570	ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ									
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ギリシア文字	2620		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
	2630	Π	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω							
	2640		α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
	2650	π	ρ	σ	τ	υ	φ	χ	ψ	ω							
ロシア文字	2720		А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н
	2730	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э
	2740	Ю	Я														
	2750		а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н
	2760	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э
	2770	ю	я														
ア	3020		亜	唖	娃	阿	哀	愛	挨	姶	逢	葵	茜	穐	悪	握	渥
	3030	旭	葦	芦	鯵	梓	圧	斡	扱	宛	姐	虻	飴	絢	綾	鮎	或
	3040	粟	袷	安	庵	按	暗	案	闇	鞍	杏						
イ	3040											以	伊	位	依	偉	囲
	3050	夷	委	威	尉	惟	意	慰	易	椅	為	畏	異	移	維	緯	胃
	3060	萎	衣	謂	違	遺	医	井	亥	域	育	郁	磯	一	壱	溢	逸
	3070	稲	茨	芋	鰯	允	印	咽	員	因	姻	引	飲	淫	胤	蔭	
	3120		院	陰	隠	韻	吋										
ウ	3120							右	宇	烏	羽	迂	雨	卯	鵜	窺	丑
	3130	碓	臼	渦	嘘	唄	欝	蔚	鰻	姥	厩	浦	瓜	閏	噂	云	運
	3140	雲															
エ	3140		荏	餌	叡	営	嬰	影	映	曳	栄	永	泳	洩	瑛	盈	穎
	3150	頴	英	衛	詠	鋭	液	疫	益	駅	悦	謁	越	閲	榎	厭	円
	3160	園	堰	奄	宴	延	怨	掩	援	沿	演	炎	焔	煙	燕	猿	縁
	3170	艶	苑	薗	遠	鉛	鴛	塩									
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
オ	3170								於	汚	甥	凹	央	奥	往	応	
	3220		押	旺	横	欧	殴	王	翁	襖	鶯	鴎	黄	岡	沖	荻	億
	3230	屋	憶	臆	桶	牡	乙	俺	卸	恩	温	穏	音				
カ	3230													下	化	仮	何
	3240	伽	価	佳	加	可	嘉	夏	嫁	家	寡	科	暇	果	架	歌	河
	3250	火	珂	禍	禾	稼	箇	花	苛	茄	荷	華	菓	蝦	課	嘩	貨
	3260	迦	過	霞	蚊	俄	峨	我	牙	画	臥	芽	蛾	賀	雅	餓	駕
	3270	介	会	解	回	塊	壊	廻	快	怪	悔	恢	懐	戒	拐	改	
	3320		魁	晦	械	海	灰	界	皆	絵	芥	蟹	開	階	貝	凱	劾
	3330	外	咳	害	崖	慨	概	涯	碍	蓋	街	該	鎧	骸	浬	馨	蛙
	3340	垣	柿	蠣	鈎	劃	嚇	各	廓	拡	攪	格	核	殻	獲	確	穫
	3350	覚	角	赫	較	郭	閣	隔	革	学	岳	楽	額	顎	掛	笠	樫
	3360	橿	梶	鰍	潟	割	喝	恰	括	活	渇	滑	葛	褐	轄	且	鰹
	3370	叶	椛	樺	鞄	株	兜	竃	蒲	釜	鎌	噛	鴨	栢	茅	萱	
	3420		粥	刈	苅	瓦	乾	侃	冠	寒	刊	勘	勧	巻	喚	堪	姦
	3430	完	官	寛	干	幹	患	感	慣	憾	換	敢	柑	桓	棺	款	歓
	3440	汗	漢	澗	灌	環	甘	監	看	竿	管	簡	緩	缶	翰	肝	艦
	3450	莞	観	諫	貫	還	鑑	間	閑	関	陥	韓	館	舘	丸	含	岸
	3460	巌	玩	癌	眼	岩	翫	贋	雁	頑	顔	願					
キ	3460												企	伎	危	喜	器
	3470	基	奇	嬉	寄	岐	希	幾	忌	揮	机	旗	既	期	棋	棄	
	3520		機	帰	毅	気	汽	畿	祈	季	稀	紀	徽	規	記	貴	起
	3530	軌	輝	飢	騎	鬼	亀	偽	儀	妓	宜	戯	技	擬	欺	犠	疑
	3540	祇	義	蟻	誼	議	掬	菊	鞠	吉	吃	喫	桔	橘	詰	砧	杵
	3550	黍	却	客	脚	虐	逆	丘	久	仇	休	及	吸	宮	弓	急	救
	3560	朽	求	汲	泣	灸	球	究	窮	笈	級	糾	給	旧	牛	去	居
	3570	巨	拒	拠	挙	渠	虚	許	距	鋸	漁	禦	魚	亨	享	京	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	3620		供	侠	僑	兇	競	共	凶	協	匡	卿	叫	喬	境	峡	強
	3630	彊	怯	恐	恭	挾	教	橋	況	狂	狭	矯	胸	脅	興	蕎	郷
	3640	鏡	響	饗	驚	仰	凝	尭	暁	業	局	曲	極	玉	桐	粁	僅
	3650	勤	均	巾	錦	斤	欣	欽	琴	禁	禽	筋	緊	芹	菌	衿	襟
	3660	謹	近	金	吟	銀											
ク	3660						九	倶	句	区	狗	玖	矩	苦	躯	駆	駈
	3670	駒	具	愚	虞	喰	空	偶	寓	遇	隅	串	櫛	釧	屑	屈	
	3720		掘	窟	沓	靴	轡	窪	熊	隈	粂	栗	繰	桑	鍬	勲	君
	3730	薫	訓	群	軍	郡											
ケ	3730						卦	袈	祁	係	傾	刑	兄	啓	圭	珪	型
	3740	契	形	径	恵	慶	慧	憩	掲	携	敬	景	桂	渓	畦	稽	系
	3750	経	継	繋	罫	茎	荊	蛍	計	詣	警	軽	頚	鶏	芸	迎	鯨
	3760	劇	戟	撃	激	隙	桁	傑	欠	決	潔	穴	結	血	訣	月	件
	3770	倹	倦	健	兼	券	剣	喧	圏	堅	嫌	建	憲	懸	拳	捲	
	3820		検	権	牽	犬	献	研	硯	絹	県	肩	見	謙	賢	軒	遣
	3830	鍵	険	顕	験	鹸	元	原	厳	幻	弦	減	源	玄	現	絃	舷
	3840	言	諺	限													
コ	3840				乎	個	古	呼	固	姑	孤	己	庫	弧	戸	故	枯
	3850	湖	狐	糊	袴	股	胡	菰	虎	誇	跨	鈷	雇	顧	鼓	五	互
	3860	伍	午	呉	吾	娯	後	御	悟	梧	檎	瑚	碁	語	誤	護	醐
	3870	乞	鯉	交	佼	侯	候	倖	光	公	功	効	勾	厚	口	向	
	3920		后	喉	坑	垢	好	孔	孝	宏	工	巧	巷	幸	広	庚	康
	3930	弘	恒	慌	抗	拘	控	攻	昂	晃	更	杭	校	梗	構	江	洪
	3940	浩	港	溝	甲	皇	硬	稿	糠	紅	紘	絞	綱	耕	考	肯	肱
	3950	腔	膏	航	荒	行	衡	講	貢	購	郊	酵	鉱	砿	鋼	閤	降
	3960	項	香	高	鴻	剛	劫	号	合	壕	拷	濠	豪	轟	麹	克	刻
	3970	告	国	穀	酷	鵠	黒	獄	漉	腰	甑	忽	惚	骨	狛	込	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	3A20		此	頃	今	困	坤	墾	婚	恨	懇	昏	昆	根	梱	混	痕
	3A30	紺	艮	魂													
サ	3A30				些	佐	叉	唆	嵯	左	差	査	沙	瑳	砂	詐	鎖
	3A40	裟	坐	座	挫	債	催	再	最	哉	塞	妻	宰	彩	才	採	栽
	3A50	歳	済	災	采	犀	砕	砦	祭	斎	細	菜	裁	載	際	剤	在
	3A60	材	罪	財	冴	坂	阪	堺	榊	肴	咲	崎	埼	碕	鷺	作	削
	3A70	咋	搾	昨	朔	柵	窄	策	索	錯	桜	鮭	笹	匙	冊	刷	
	3B20		察	拶	撮	擦	札	殺	薩	雑	皐	鯖	捌	錆	鮫	皿	晒
	3B30	三	傘	参	山	惨	撒	散	桟	燦	珊	産	算	纂	蚕	讃	賛
	3B40	酸	餐	斬	暫	残											
シ	3B40						仕	仔	伺	使	刺	司	史	嗣	四	士	始
	3B50	姉	姿	子	屍	市	師	志	思	指	支	孜	斯	施	旨	枝	止
	3B60	死	氏	獅	祉	私	糸	紙	紫	肢	脂	至	視	詞	詩	試	誌
	3B70	諮	資	賜	雌	飼	歯	事	似	侍	児	字	寺	慈	持	時	
	3C20		次	滋	治	爾	璽	痔	磁	示	而	耳	自	蒔	辞	汐	鹿
	3C30	式	識	鴫	竺	軸	宍	雫	七	叱	執	失	嫉	室	悉	湿	漆
	3C40	疾	質	実	蔀	篠	偲	柴	芝	屡	蕊	縞	舎	写	射	捨	赦
	3C50	斜	煮	社	紗	者	謝	車	遮	蛇	邪	借	勺	尺	杓	灼	爵
	3C60	酌	釈	錫	若	寂	弱	惹	主	取	守	手	朱	殊	狩	珠	種
	3C70	腫	趣	酒	首	儒	受	呪	寿	授	樹	綬	需	囚	収	周	
	3D20		宗	就	州	修	愁	拾	洲	秀	秋	終	繍	習	臭	舟	蒐
	3D30	衆	襲	讐	蹴	輯	週	酋	酬	集	醜	什	住	充	十	従	戎
	3D40	柔	汁	渋	獣	縦	重	銃	叔	夙	宿	淑	祝	縮	粛	塾	熟
	3D50	出	術	述	俊	峻	春	瞬	竣	舜	駿	准	循	旬	楯	殉	淳
	3D60	準	潤	盾	純	巡	遵	醇	順	処	初	所	暑	曙	渚	庶	緒
	3D70	署	書	薯	藷	諸	助	叙	女	序	徐	恕	鋤	除	傷	償	
	3E20		勝	匠	升	召	哨	商	唱	嘗	奨	妾	娼	宵	将	小	少
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	3E30	尚	庄	床	廠	彰	承	抄	招	掌	捷	昇	昌	昭	晶	松	梢
	3E40	樟	樵	沼	消	渉	湘	焼	焦	照	症	省	硝	礁	祥	称	章
	3E50	笑	粧	紹	肖	菖	蔣	蕉	衝	裳	訟	証	詔	詳	象	賞	醬
	3E60	鉦	鍾	鐘	障	鞘	上	丈	丞	乗	冗	剰	城	場	壌	嬢	常
	3E70	情	擾	条	杖	浄	状	畳	穣	蒸	譲	醸	錠	嘱	埴	飾	
	3F20		拭	植	殖	燭	織	職	色	触	食	蝕	辱	尻	伸	信	侵
	3F30	唇	娠	寝	審	心	慎	振	新	晋	森	榛	浸	深	申	疹	真
	3F40	神	秦	紳	臣	芯	薪	親	診	身	辛	進	針	震	人	仁	刃
	3F50	塵	壬	尋	甚	尽	腎	訊	迅	陣	靱						
ス	3F50											笥	諏	須	酢	図	厨
	3F60	逗	吹	垂	帥	推	水	炊	睡	粋	翠	衰	遂	酔	錐	錘	随
	3F70	瑞	髄	崇	嵩	数	枢	趨	雛	据	杉	椙	菅	頗	雀	裾	
	4020		澄	摺	寸												
セ	4020					世	瀬	畝	是	凄	制	勢	姓	征	性	成	政
	4030	整	星	晴	棲	栖	正	清	牲	生	盛	精	聖	声	製	西	誠
	4040	誓	請	逝	醒	青	静	斉	税	脆	隻	席	惜	戚	斥	昔	析
	4050	石	積	籍	績	脊	責	赤	跡	蹟	碩	切	拙	接	摂	折	設
	4060	窃	節	説	雪	絶	舌	蟬	仙	先	千	占	宣	専	尖	川	戦
	4070	扇	撰	栓	栴	泉	浅	洗	染	潜	煎	煽	旋	穿	箭	線	
	4120		繊	羨	腺	舛	船	薦	詮	賤	践	選	遷	銭	銑	閃	鮮
	4130	前	善	漸	然	全	禅	繕	膳	糎							
ソ	4130										噌	塑	岨	措	曾	曽	楚
	4140	狙	疏	疎	礎	祖	租	粗	素	組	蘇	訴	阻	遡	鼠	僧	創
	4150	双	叢	倉	喪	壮	奏	爽	宋	層	匝	惣	想	捜	掃	挿	搔
	4160	操	早	曹	巣	槍	槽	漕	燥	争	痩	相	窓	糟	総	綜	聡
	4170	草	荘	葬	蒼	藻	装	走	送	遭	鎗	霜	騒	像	増	憎	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	4220		臓	蔵	贈	造	促	側	則	即	息	捉	束	測	足	速	俗
	4230	属	賊	族	続	卒	袖	其	揃	存	孫	尊	損	村	遜		
タ	4230															他	多
	4240	太	汰	詑	唾	堕	妥	惰	打	柁	舵	楕	陀	駄	騨	体	堆
	4250	対	耐	岱	帯	待	怠	態	戴	替	泰	滞	胎	腿	苔	袋	貸
	4260	退	逮	隊	黛	鯛	代	台	大	第	醍	題	鷹	滝	瀧	卓	啄
	4270	宅	托	択	拓	沢	濯	琢	託	鐸	濁	諾	茸	凧	蛸	只	
	4320		叩	但	達	辰	奪	脱	巽	竪	辿	棚	谷	狸	鱈	樽	誰
	4330	丹	単	嘆	坦	担	探	旦	歎	淡	湛	炭	短	端	箪	綻	耽
	4340	胆	蛋	誕	鍛	団	壇	弾	断	暖	檀	段	男	談			
チ	4340														値	知	地
	4350	弛	恥	智	池	痴	稚	置	致	蜘	遅	馳	築	畜	竹	筑	蓄
	4360	逐	秩	窒	茶	嫡	着	中	仲	宙	忠	抽	昼	柱	注	虫	衷
	4370	註	酎	鋳	駐	樗	瀦	猪	苧	著	貯	丁	兆	凋	喋	寵	
	4420		帖	帳	庁	弔	張	彫	徴	懲	挑	暢	朝	潮	牒	町	眺
	4430	聴	脹	腸	蝶	調	諜	超	跳	銚	長	頂	鳥	勅	捗	直	朕
	4440	沈	珍	賃	鎮	陳											
ツ	4440						津	墜	椎	槌	追	鎚	痛	通	塚	栂	掴
	4450	槻	佃	漬	柘	辻	蔦	綴	鍔	椿	潰	坪	壷	嬬	紬	爪	吊
	4460	釣	鶴														
テ	4460			亭	低	停	偵	剃	貞	呈	堤	定	帝	底	庭	廷	弟
	4470	悌	抵	挺	提	梯	汀	碇	禎	程	締	艇	訂	諦	蹄	逓	
	4520		邸	鄭	釘	鼎	泥	摘	擢	敵	滴	的	笛	適	鏑	溺	哲
	4530	徹	撤	轍	迭	鉄	典	填	天	展	店	添	纒	甜	貼	転	顛
	4540	点	伝	殿	澱	田	電										
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ト	4540							兎	吐	堵	塗	妬	屠	徒	斗	杜	渡
	4550	登	菟	賭	途	都	鍍	砥	礪	努	度	土	奴	怒	倒	党	冬
	4560	凍	刀	唐	塔	塘	套	宕	島	嶋	悼	投	搭	東	桃	檮	棟
	4570	盗	淘	湯	濤	灯	燈	当	痘	禱	等	答	筒	糖	統	到	
	4620		董	蕩	藤	討	謄	豆	踏	逃	透	鐙	陶	頭	騰	闘	働
	4630	動	同	堂	導	憧	撞	洞	瞳	童	胴	萄	道	銅	峠	鴇	匿
	4640	得	徳	瀆	特	督	禿	篤	毒	独	読	栃	橡	凸	突	椴	届
	4650	鳶	苫	寅	酉	瀞	噸	屯	惇	敦	沌	豚	遁	頓	呑	曇	鈍
ナ	4660	奈	那	内	乍	凪	薙	謎	灘	捺	鍋	楢	馴	縄	畷	南	楠
	4670	軟	難	汝													
ニ	4670				二	尼	弐	邇	匂	賑	肉	虹	廿	日	乳	入	
	4720		如	尿	韮	任	妊	忍	認								
ヌ	4720									濡							
ネ	4720										禰	祢	寧	葱	猫	熱	年
	4730	念	捻	撚	燃	粘											
ノ	4730						乃	廼	之	埜	囊	悩	濃	納	能	脳	膿
	4740	農	覗	蚤													
ハ	4740				巴	把	播	覇	杷	波	派	琶	破	婆	罵	芭	馬
	4750	俳	廃	拝	排	敗	杯	盃	牌	背	肺	輩	配	倍	培	媒	梅
	4760	楳	煤	狽	買	売	賠	陪	這	蠅	秤	矧	萩	伯	剝	博	拍
	4770	柏	泊	白	箔	粕	舶	薄	迫	曝	漠	爆	縛	莫	駁	麦	
	4820		函	箱	硲	箸	肇	筈	櫨	幡	肌	畑	畠	八	鉢	潑	発
	4830	醱	髪	伐	罰	抜	筏	閥	鳩	噺	塙	蛤	隼	伴	判	半	反
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	4840	叛	帆	搬	斑	板	氾	汎	版	犯	班	畔	繁	般	藩	販	範
	4850	釆	煩	頒	飯	挽	晩	番	盤	磐	蕃	蛮					
ヒ	4850												匪	卑	否	妃	庇
	4860	彼	悲	扉	批	披	斐	比	泌	疲	皮	碑	秘	緋	罷	肥	被
	4870	誹	費	避	非	飛	樋	簸	備	尾	微	枇	毘	琵	眉	美	
	4920		鼻	柊	稗	匹	疋	髭	彦	膝	菱	肘	弼	必	畢	筆	逼
	4930	桧	姫	媛	紐	百	謬	俵	彪	標	氷	漂	瓢	票	表	評	豹
	4940	廟	描	病	秒	苗	錨	鋲	蒜	蛭	鰭	品	彬	斌	浜	瀕	貧
	4950	賓	頻	敏	瓶												
フ	4950					不	付	埠	夫	婦	富	冨	布	府	怖	扶	敷
	4960	斧	普	浮	父	符	腐	膚	芙	譜	負	賦	赴	阜	附	侮	撫
	4970	武	舞	葡	蕪	部	封	楓	風	葺	蕗	伏	副	復	幅	服	
	4A20		福	腹	複	覆	淵	弗	払	沸	仏	物	鮒	分	吻	噴	墳
	4A30	憤	扮	焚	奮	粉	糞	紛	雰	文	聞						
ヘ	4A30											丙	併	兵	塀	幣	平
	4A40	弊	柄	並	蔽	閉	陛	米	頁	僻	壁	癖	碧	別	瞥	蔑	箆
	4A50	偏	変	片	篇	編	辺	返	遍	便	勉	娩	弁	鞭			
ホ	4A50														保	舗	鋪
	4A60	圃	捕	歩	甫	補	輔	穂	募	墓	慕	戊	暮	母	簿	菩	倣
	4A70	俸	包	呆	報	奉	宝	峰	峯	崩	庖	抱	捧	放	方	朋	
	4B20		法	泡	烹	砲	縫	胞	芳	萌	蓬	蜂	褒	訪	豊	邦	鋒
	4B30	飽	鳳	鵬	乏	亡	傍	剖	坊	妨	帽	忘	忙	房	暴	望	某
	4B40	棒	冒	紡	肪	膨	謀	貌	貿	鉾	防	吠	頬	北	僕	卜	墨
	4B50	撲	朴	牧	睦	穆	釦	勃	没	殆	堀	幌	奔	本	翻	凡	盆
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
マ	4B60	摩	磨	魔	麻	埋	妹	昧	枚	毎	哩	槙	幕	膜	枕	鮪	柾
	4B70	鱒	桝	亦	俣	又	抹	末	沫	迄	儘	繭	麿	万	慢	満	
	4C20		漫	蔓													
ミ	4C20				味	未	魅	巳	箕	岬	密	蜜	湊	蓑	稔	脈	妙
	4C30	粍	民	眠													
ム	4C30				務	夢	無	牟	矛	霧	鵡	椋	婿	娘			
メ	4C30														冥	名	命
	4C40	明	盟	迷	銘	鳴	姪	牝	滅	免	棉	綿	緬	面	麺		
モ	4C40															摸	模
	4C50	茂	妄	孟	毛	猛	盲	網	耗	蒙	儲	木	黙	目	杢	勿	餅
	4C60	尤	戻	籾	貰	問	悶	紋	門	匁							
ヤ	4C60										也	冶	夜	爺	耶	野	弥
	4C70	矢	厄	役	約	薬	訳	躍	靖	柳	薮	鑓					
ユ	4C70												愉	愈	油	癒	
	4D20		諭	輸	唯	佑	優	勇	友	宥	幽	悠	憂	揖	有	柚	湧
	4D30	涌	猶	猷	由	祐	裕	誘	遊	邑	郵	雄	融	夕			
ヨ	4D30														予	余	与
	4D40	誉	輿	預	傭	幼	妖	容	庸	揚	揺	擁	曜	楊	様	洋	溶
	4D50	熔	用	窯	羊	耀	葉	蓉	要	謡	踊	遥	陽	養	慾	抑	欲
	4D60	沃	浴	翌	翼	淀											
ラ	4D60						羅	螺	裸	来	莱	頼	雷	洛	絡	落	酪
	4D70	乱	卵	嵐	欄	濫	藍	蘭	覧								
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
リ	4D70									利	吏	履	李	梨	理	璃	
	4E20		痢	裏	裡	里	離	陸	律	率	立	葎	掠	略	劉	流	溜
	4E30	琉	留	硫	粒	隆	竜	龍	侶	慮	旅	虜	了	亮	僚	両	凌
	4E40	寮	料	梁	涼	猟	療	瞭	稜	糧	良	諒	遼	量	陵	領	力
	4E50	緑	倫	厘	林	淋	燐	琳	臨	輪	隣	鱗	麟				
ル	4E50													瑠	塁	涙	累
	4E60	類															
レ	4E60		令	伶	例	冷	励	嶺	怜	玲	礼	苓	鈴	隷	零	霊	麗
	4E70	齢	暦	歴	列	劣	烈	裂	廉	恋	憐	漣	煉	簾	練	聯	
	4F20		蓮	連	錬												
ロ	4F20					呂	魯	櫓	炉	賂	路	露	労	婁	廊	弄	朗
	4F30	楼	榔	浪	漏	牢	狼	籠	老	聾	蝋	郎	六	麓	禄	肋	録
	4F40	論															
ワ	4F40		倭	和	話	歪	賄	脇	惑	枠	鷲	亙	亘	鰐	詫	藁	蕨
	4F50	椀	湾	碗	腕												
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

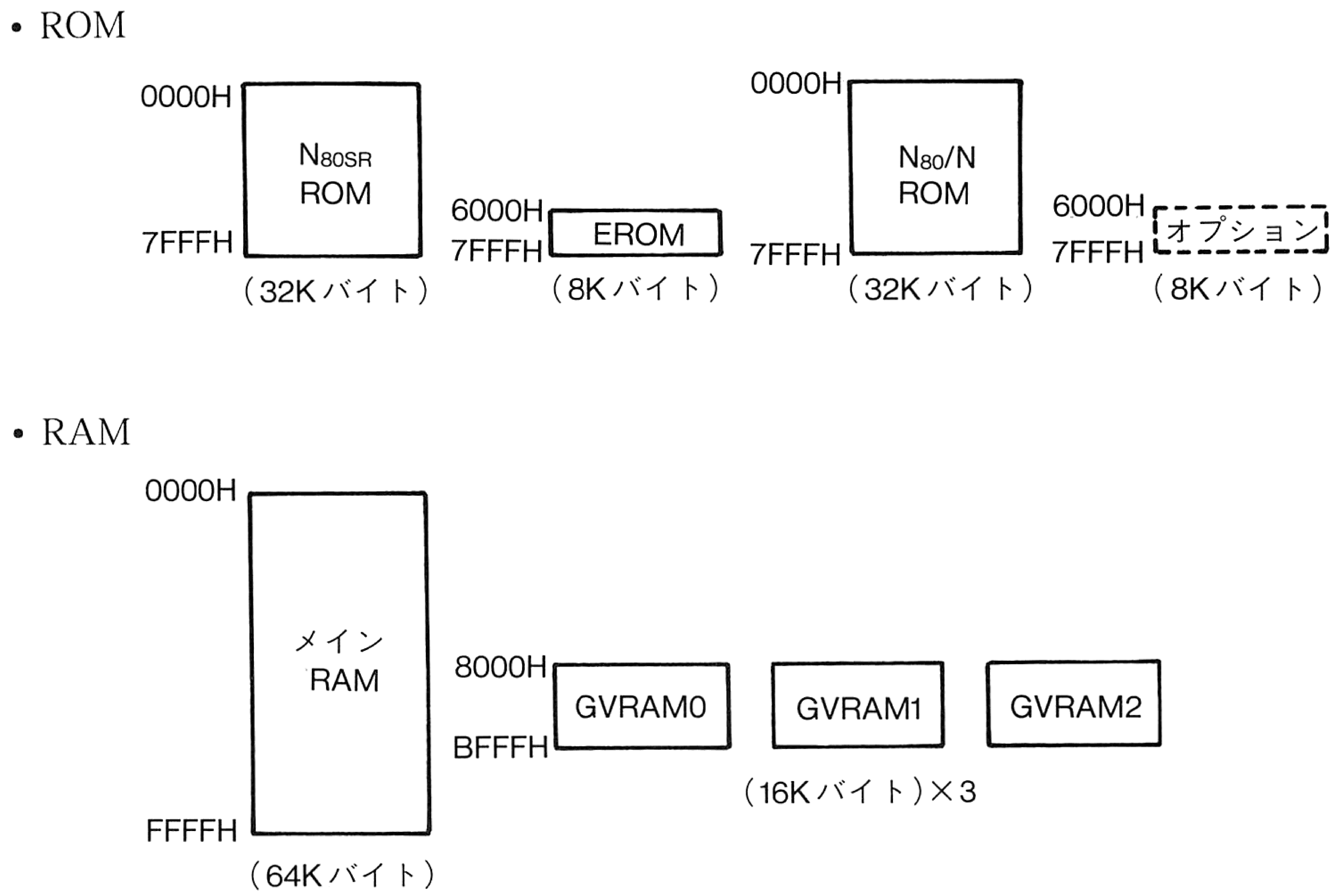
資料12　誘導関数

N_{80SR}-BASICが“組み込み”関数として用意していない三角関数のうち，いくつかは，用意された関数を使って作り出すことができます．以下の数式を参考にしてください(誤差の範囲に注意が必要です)．

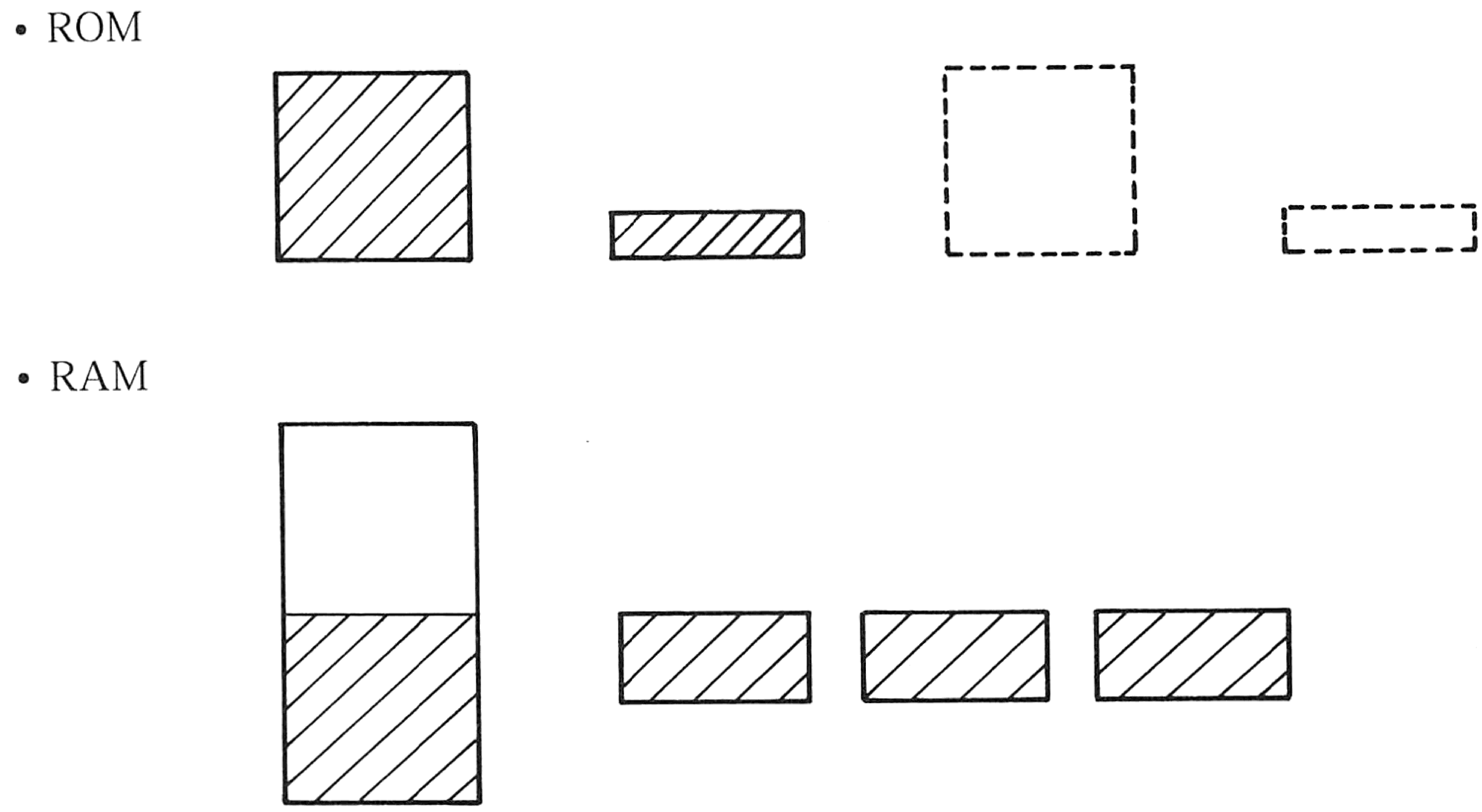
目的とする関数	組み込み関数からの誘導式
セカント	SEC(X)=1/COS(X)
コセカント	COSEC(X)=1/SIN(X)
コタンジェント	COT(X)=1/TAN(X)
アークサイン	ARCSIN(X)=ATN(X/SQR(−X*X+1))
アークコサイン	ARCCOS(X)=−ATN(X/SQR(−X*X+1))+1.5708
アークセカント	ARCSEC(X)=ATN(SQR(X*X−1))+(SGN(X)−1)*1.5708
アークコセカント	ARCCOSEC(X)=ATN(1/SQR(X*X−1))+(SGN(X)−1)*1.5708
アークコタンジェント	ARCCOT(X)=−ATN(X)+1.5708
ハイパーボリック・サイン	SINH(X)=(EXP(X)−EXP(−X))/2
ハイパーボリック・コサイン	COSH(X)=EXP(−X)/2
ハイパーボリック・タンジェント	TANH(X)=−EXP(−X)/(EXP(X)+EXP(−X))*2+1
ハイパーボリック・セカント	SECH(X)=2/(EXP(X)+EXP(−X))
ハイパーボリック・コセカント	COSECH(X)=2/(EXP(X)−EXP(−X))
ハイパーボリック・コタンジェント	COTH(X)=EXP(−X)/(EXP(X)−EXP(−X))*2+1
ハイパーボリック・アークサイン	ARCSINH(X)=LOG(X+SQR(X*X+1))
ハイパーボリック・アークコサイン	ARCCOSH(X)=LOG(X+SQR(X*X−1))
ハイパーボリック・アークタンジェント	ARCTANH(X)=LOG((1+X)/(1−X))/2
ハイパーボリック・アークセカント	ARCSECH(X)=LOG((SQR(−X*X+1)+1)/X)
ハイパーボリック・アークコセカント	ARCCOSECH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
ハイパーボリック・アークコタンジェント	ARCCOTH(X)=LOG((X+1)/(X−1))/2
常用対数	LOG_{10}(X)=LOG(X)/LOG(10)

資料13　メモリマップ

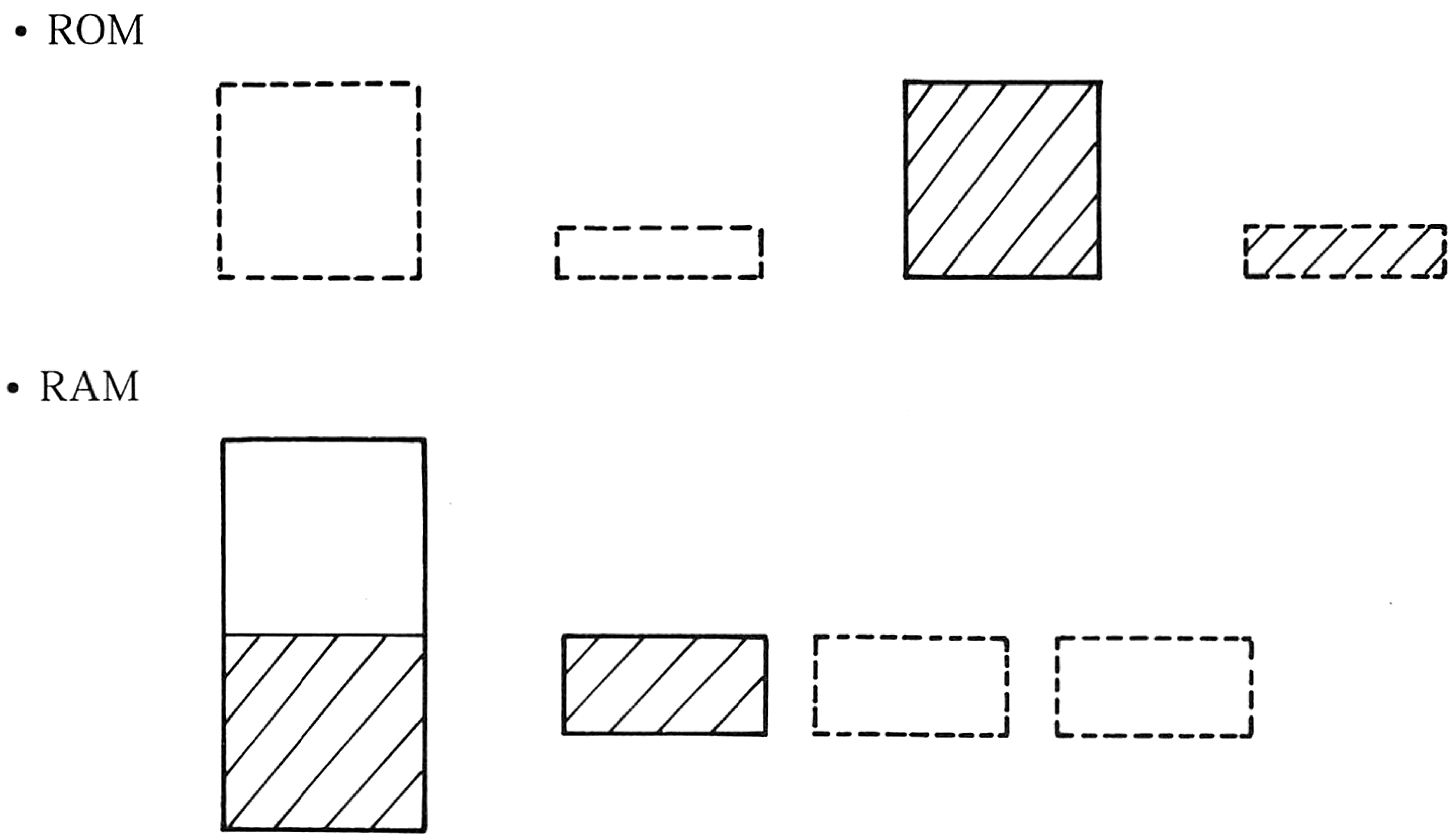
PC-8001MKⅡSRのメモリマップは次のようになっています.



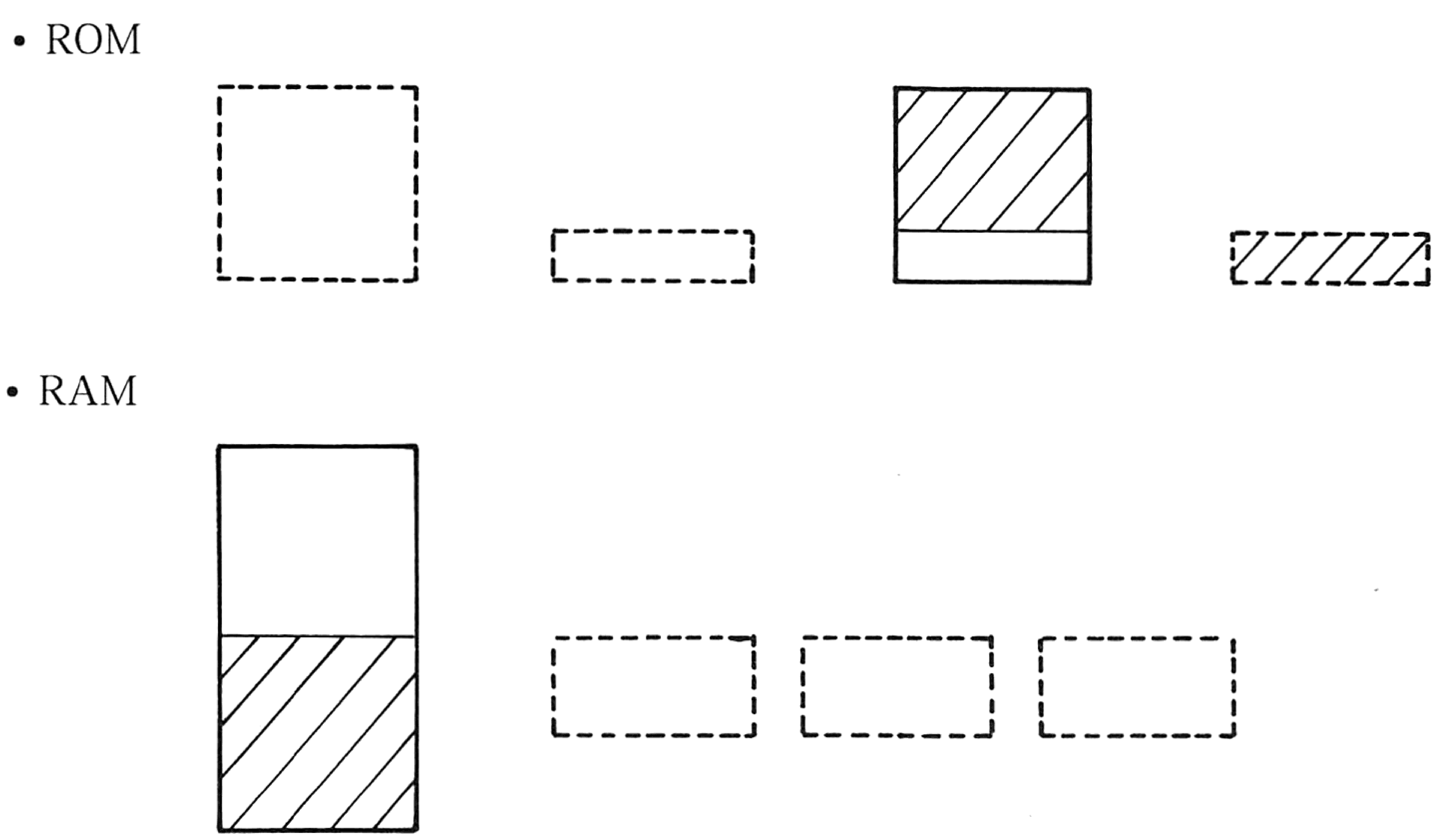
PC-8001MKⅡSRの全メモリ



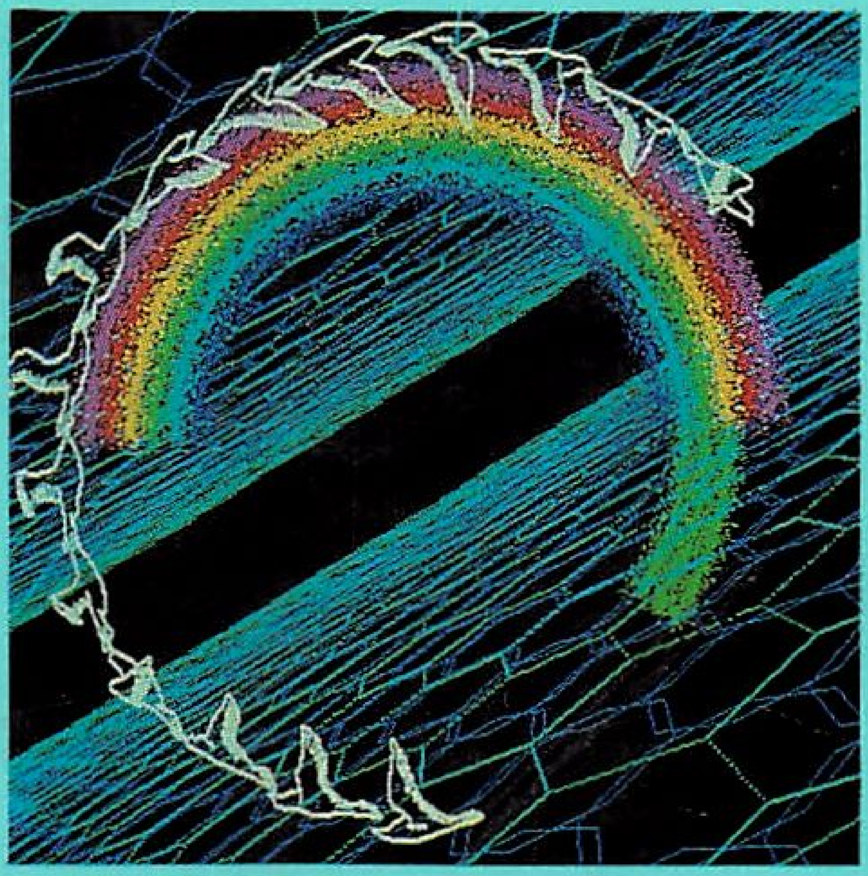
N80SR-BASICで使われるメモリ



N_{80}-BASICで使われるメモリ



N-BASICで使われるメモリ



NEC 日本電気株式会社・日本電気ホームエレクトロニクス株式会社

78115291